A RECURSIVE GRAPH BIPARTITIONING ALGORITHM BY VERTEX SEPARATORS WITH FIXED VERTICES FOR PERMUTING SPARSE MATRICES INTO BLOCK DIAGONAL FORM WITH OVERLAP

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Seher Acer

September, 2011

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Cevdet Aykanat (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Hakan Ferhatosmanoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Oya Ekin Karaşan

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

A RECURSIVE GRAPH BIPARTITIONING ALGORITHM BY VERTEX SEPARATORS WITH FIXED VERTICES FOR PERMUTING SPARSE MATRICES INTO BLOCK DIAGONAL FORM WITH OVERLAP

Seher Acer
M.S. in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
September, 2011

Solving sparse system of linear equations Ax=b using preconditioners can be efficiently parallelized using graph partitioning tools. In this thesis, we investigate the problem of permuting a sparse matrix into a block diagonal form with overlap which is to be used in the parallelization of the multiplicative schwarz preconditioner. A matrix is said to be in block diagonal form with overlap if the diagonal blocks may overlap. In order to formulate this permutation problem as a graph-theoretical problem, we introduce a restricted version of the graph partitioning by vertex separator problem (GPVS), where the objective is to find a vertex partition whose parts are only connected by a vertex separator. The modified problem, we refer as ordered GPVS problem (oGPVS), is restricted such that the parts should exhibit an ordered form where the consecutive parts can only be connected by a separator.

The existing graph partitioning tools are unable to solve the oGPVS problem. Thus, we present a recursive graph bipartitioning algorithm by vertex separators together with a novel vertex fixation scheme so that a GPVS tool supporting fixed vertices can effectively and efficiently be utilized. We also theoretically verified the correctness of the proposed approach devising a necessary and sufficient condition to the feasibility of a oGPVS solution. Experimental results on a wide range of matrices confirm the validity of the proposed approach.

Keywords: graph partitioning by vertex separator, combinatorial scientific computing, parallel computing, block diagonal form with overlap.

ÖZET

SEYREK MATRİSLERİN ÖRTÜŞEN BLOK KÖŞEGEN BİÇİME DÜZENLENMESİ İÇİN DÜĞÜM AYIRACI VE SABİT DÜĞÜMLERİ KULLANAN ÖZYİNELİ BİR ÇİZGE BÖLÜMLEME ALGORİTMASI

Seher Acer
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Eylül, 2011

Ax=b şeklindeki seyrek doğrusal denklem sistemlerinin ön hazırlık kullanılarak çözümü çizge bölümleme araçları kullanılarak etkili ve verimli bir biçimde koşut hesaplamasına uygun hale getirilebilir. Bu tez çalışmasında, çarpımsal schwarz ön hazırlayıcısının koşut hesaplanmasında kullanılmak üzere bir seyrek matrisin örtüşen blok köşegen biçimine yeniden düzenlenmesi problemi incelenmektedir. Ardışık köşegen blokları örtüşen blok köşegen matrislere örtüşen blok köşegen matrisler denir. Bu yeniden düzenleme probleminin çizge kuramı kullanılarak ifade edilebilmesi için Düğüm Ayıracı ile Çizge Bölümleme (DAÇB) probleminin kısıtlı bir çeşidi olan sıralı DAÇB (sDAÇB) problemi tanıtılmaktadır. sDAÇB probleminde amaç iki ardışık düğüm bölümünün sadece bir düğüm ayıracı ile bağlanabildiği sıralı bir düğüm bölümlemesi bulmaktır.

Varolan çizge bölümleme araçları sDAÇB problemini çözememektedirler. Bu nedenle, bu tez çalışmasında, düğüm ayıraçlarını ve yeni bir düğüm sabitleme düzenini kullanan özyineli bir çizge bölümleme algoritması önerilmektedir. Bu algoritma ile sabit düğümleri destekleyen bir DAÇB aracı etkili ve verimli bir şekilde kullanılabilmektedir. Ayrıca, bir sDAÇB çözümünün uygulanabilirliği için yeterli ve gerekli koşul incelenerek önerilen yaklaşım kuramsal olarak doğrulanmıştır. Çeşitli matrisler üzerinde yapılan deneylerin sonuçları önerilen yaklaşımın geçerliliğini doğrulamaktadır.

Anahtar sözcükler: düğüm ayıracı ile çizge bölümleme, kombinatoriyal bilimsel hesaplama, koşut hesaplama, örtüşen blok köşegen matris.

Acknowledgement

I would like to express my deepest gratitude to my supervisor Prof. Dr. Cevdet Aykanat for guidance, suggestions, and invaluable encouragement throughout the development of this thesis.

I owe special thanks to Enver Kayaaslan, who contributed continuously through the design and development of the studies we explain in this thesis.

I am grateful to Assoc. Prof. Dr. Hakan Ferhatosmanoğlu and Assoc. Prof. Dr. Oya Ekin Karaşan for reading and commenting on the thesis.

I am grateful to all of my friends and colleagues for their moral and intellectual support during my studies, especially to Özlem, Damla, Elif, Merve and my officemates, Enver, Şükrü, Çağrı, Zeynep, Mustafa, Emre and Bengü.

I would like to thank to my family, especially to my sister, for their persistent support, encouragement, understanding and love.

Finally, very special thanks goes to Hadi Eloy, who has been my side in every aspect of life with his endless love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graph/hypergraph partitioning is commonly used to distribute workload for an efficient parallelization of solving a sparse system of linear equations $Ax = b$. Roughly speaking, the vertices represent the data and the computations, and the (hyper)edges represent dependencies of the computations into the data. For a parallel system, partitioning the vertices into K parts corresponds to partitioning the data and computations among K processors by assigning the data associated with each part to a unique processor. For an efficient parallelism, the workload performed by each processor should be almost the same and the communication volume among the processors should be minimized. Equivalently, the objective of the graph partitioning problem is to minimize the number of edges that connect different parts while maintaining balance on the part weights. Output of the graph partitioning, i.e., partition of vertices, is used to permute the rows and columns of A such that the permuted matrix exhibits a block diagonal form where the data and the computations of each block are assigned to a different processor. A number of state-of-the-art graph/hypergraph partitioning tools such as Chaco [16], MeTiS [20], PaToH [8], Scotch[24], and Zoltan [4] are publicly available and widely used in many applications.

One possible approach to achieve an effective parallelism is to permute the matrix A into a doubly bordered (DB) block diagonal form which is used in many applications such as domain decomposition-based solvers [13, 23, 26], preconditioned iterative methods [3], and hybrid solvers [21, 28]. The DB block diagonal form is a variant

of the block diagonal form where off-diagonal nonzeros reside only in the bottommost row and the leftmost column stripes. Permuting a matrix into the DB block diagonal form is a well-known problem, and the graph partitioning by vertex separator (GPVS) problem is utilized in a typical solution of this permutation problem.

The GPVS problem is a well-known variant of the graph partitioning problem where the parts can only be connected by a set of vertices, called vertex separator. That is, the removal of the separator vertices decomposes the graph into K subgraphs such that the vertex set of each subgraph corresponds to a part in the partition. The objective of the GPVS problem is to minimize the separator size while maintaining a balance on the part weights. The GPVS problem, which is widely used in nested-dissection-based low-fill orderings for factorization of symmetric sparse matrices, is known to be NP-hard [5].

In this thesis, our target problem, which we refer to as A-to-A_{BDO} permutation problem, is to symmetrically permute rows and columns of an $N \times N$ structurally symmetric sparse matrix A into a K-way block diagonal (BDO) form A^π with overlap:

$$A^\pi = PAP^T = A_{BDO} = \begin{bmatrix} A_{1,1} & A_{1,2} & & & & & \\ A_{1,2}^T & C_{1,1} & A_{2,1} & C_{1,2} & & & \\ & A_{2,1}^T & A_{2,2} & A_{2,3} & & & \\ & C_{1,2}^T & A_{2,3}^T & C_{2,2} & \cdots & & \\ & & & \vdots & \ddots & & \\ & & & & & C_{K-1,K-1} & A_{K,K-1} \\ & & & & & A_{K,K-1}^T & A_{K,K} \end{bmatrix}, \tag{1.1}$$

Here, P denotes an $N \times N$ permutation matrix. The BDO form contains K diagonal blocks $D_1, D_2, \ldots, D_K$, where

$$D_k = \begin{bmatrix} C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\ A_{k,k-1}^T & A_{k,k} & A_{k,k+1} \\ C_{k-1,k}^T & A_{k,k+1}^T & C_{k,k} \end{bmatrix} \text{ for } k = 2, 3, \ldots, K-1, \tag{1.2}$$

$$D_1 = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{1,2}^T & C_{1,1} \end{bmatrix}, \quad D_K = \begin{bmatrix} C_{K-1,K-1} & A_{K,K-1} \\ A_{K,K-1}^T & A_{K,K} \end{bmatrix}. \tag{1.3}$$

In (1.2), $C_{k,k}$ denotes the coupling diagonal block where the successive kth and $(k+1)$th diagonal blocks D_k and D_{k+1} overlap. The diagonal blocks D_k's and the coupling diagonal blocks $C_{k,k}$'s for $k = 1, 2, \ldots, K$ are square submatrices as well as the matrix A. However, D_k's and $C_{k,k}$'s may consist of varying numbers of rows/columns through $k = 1, 2, \ldots, K$. Note that A_{BDO} is structurally symmetric since a symmetric permutation is applied on the symmetric matrix A. Figure 1.1 displays a better visualization of the BDO form of the matrix A. The objective of the A-to-A_{BDO} permutation is to minimize the sum of the number of rows/columns of the coupling diagonal blocks, whereas the permutation constraint is to maintain balance on the nonzero counts of the diagonal blocks.

The A-to-A_{BDO} permutation problem arises in the parallelization of the multiplicative schwarz preconditioner given in [18]. In this parallelization, each diagonal block D_k of the permuted matrix A_{BDO} together with the associated computations are assigned to a distinct processor k. The permutation objective of minimizing the sum of the number of rows/columns of the coupling diagonal blocks corresponds to minimizing the total communication volume of the parallel system [18]. The permutation objective also corresponds to minimizing the upper bound on the number of iterations of the solver using multiplicative schwarz preconditioner [19], since it is proven that the sum of the number of rows/columns of the coupling diagonal blocks is an upper bound on the number of iterations to convergence. The permutation constraint of maintaining balance on the nonzero counts of the diagonal blocks relates to maintaining balance on the computational loads of processors during the iterations.

The contributions of this thesis can be considered as three-fold:

1. *Defining the ordered GPVS (oGPVS) problem:* We define the oGPVS problem, which is a variant of the GPVS problem. For this purpose, we also define a special form of vertex separator, namely *ordered Vertex Separator (oVS)*, which is to be used in the oGPVS problem definition.

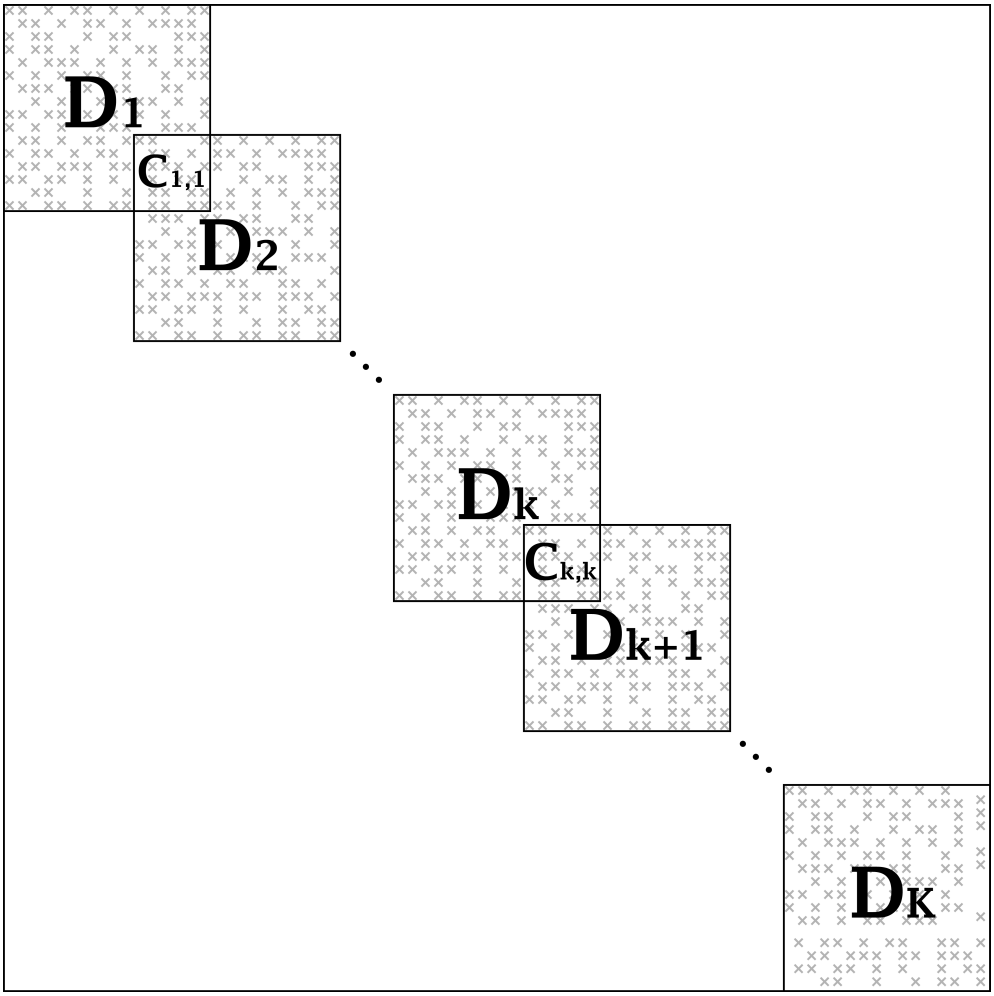

Figure 1.1: Block diagonal form with overlap

2. *Formulating the A-to-A_{BDO} permutation problem as a K-way oGPVS problem:* We show how the rows/columns of diagonal blocks D_k's and coupling diagonal blocks $C_{k,k}$'s in BDO form can be decoded by the vertices of the parts and the separator of the oVS structure. We also show the one-to-one correspondence between the objectives of A-to-A_{BDO} permutation problem and oGPVS problem, as well as the relation between the constraints of these two problems.

3. *Proposing a recursive bipartitioning (RB) based algorithm to solve the oGPVS problem:* Since existing graph partitioning tools are unable to solve the oGPVS problem, we show how the RB paradigm, which is successively and commonly used for K-way graph/hypergraph partitioning, can be utilized for solving the

oGPVS problem. For this purpose, we propose a left-to-right bipartitioning approach together with a novel vertex fixation scheme so that existing 2-way GPVS tools that support fixed vertices can effectively and efficiently be utilized in the RB framework.

The rest of the thesis is organized as follows. Related work and a detailed explanation of a previous work on the same problem is provided in Chapter 2. Chapter 3 provides a background information. The oGPVS problem formulation is presented in Chapter 4. Chapter 5 presents and discusses the RB-based algorithm proposed for solving the oGPVS problem. Implementation details and experimental results are given in Chapter 6. Finally, Chapter 7 concludes the thesis.

Chapter 2

Related Work

Block tridiagonalization and block diagonalization with overlap are closely related problems where block tridiagonalization can be considered as a special case of block diagonalization with overlap. Block tridiagonal (BT) form of a matrix A has the same structure with BDO form except that the off-diagonal submatrices $C_{k-1,k}^T$ and $C_{k-1,k}$ of each diagonal block D_k are zero. In A-to-A_{BT} permutation problem, one of the objectives is to maximize the number of blocks while maintaining a balance on the sizes of the blocks. A partitioning approach resulting in a block tridiagonal form is proposed in [14], which uses a one-way dissection and quotient tree algorithms. Another block tridiagonalization method is proposed in [27], which is to be used in a physical application, called coherent charge transport. A-to-A_{BT} and A-to-A_{BDO} permutation problems may also have a number of common steps during their solutions such as finding a pseudo-peripheral vertex and computing a level structure on the standard graph representation of A.

To our knowledge, the A-to-A_{BDO} permutation problem has only been addressed in a recent work by Kahou et al. [17]. In this work, they propose a bottom-up graph partitioning algorithm on the standard graph representation G of A, which consists of the steps explained in the rest of this chapter. Since this proposed algorithm finds a partition in a bottom-up manner and iteratively refines it, decisions of the algorithm are based on the local information. Hence, a new method which makes decisions based on the global information is needed for this permutation problem. For this purpose, we

propose a top-down partitioning algorithm which makes use of the global information and makes decisions accordingly.

Kahou's graph partitioning algorithm for A-to-A_{BDO} permutation problem has 6 basic steps which can be explained as follows:

1. *Finding a pseudo-peripheral vertex of G:* A peripheral vertex in a graph of diameter d is defined as a vertex that has distance d from some other vertex, that is, a vertex that achieves the diameter. Since finding a peripheral vertex in a graph is a hard problem, they use a pseudo-peripheral node finder algorithm, described in [15], to find a pseudo-peripheral vertex v_0.

2. *Constructing a level structure T of G rooted at v_0:* The level structure T rooted at v_0, which can be viewed as a tree, is a partition of the vertices of G according to their distances to v_0. Formally, $T = \{L_0, L_1, L_2, \ldots, L_\ell\}$ where $L_i = \{v_i : \delta(v_i, v_0) = i\}$ for $i = 1, 2, \ldots, \ell$. Here, $\delta(v_x, v_y)$ denotes the distance between vertex v_x and vertex v_y in the corresponding graph. Breadth-First Search (BFS), which is a very well known searching algorithm on graphs, is used to construct this level structure. Note that vertices in L_i can only be adjacent to the vertices in L_{i-1} and L_{i+1} for $i = 0, 1, \ldots, \ell$. Figure 2.1 displays an example level structure of length $\ell = 6$ rooted at vertex v_0. If the length of the level set T is smaller than the number K of the desired parts, then it is not possible to partition G into K parts.

3. *Gathering an initial partition P_0 of vertices to K parts from the level structure T:* The obtained level structure T is considered as a chain of tasks where each level set L_i is simply a task and the task weight $w(L_i)$ is defined as the sum of the degrees of the vertices in L_i. Thus, partitioning level structure T into K parts corresponds to finding a sequence of delimiters $\tau_1, \tau_2, \ldots, \tau_{K-1}$ while maintaining load balancing such that the tasks residing between two consecutive delimiters form a part. They use chains-on-chains partitioning [25] algorithm on this chain to find the delimiters and so the initial partition $P_0 = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$. In the initial partition P_0, each part $\mathcal{V}_i$ contains one or more consecutive levels so that all inter-part edges are confined to be between

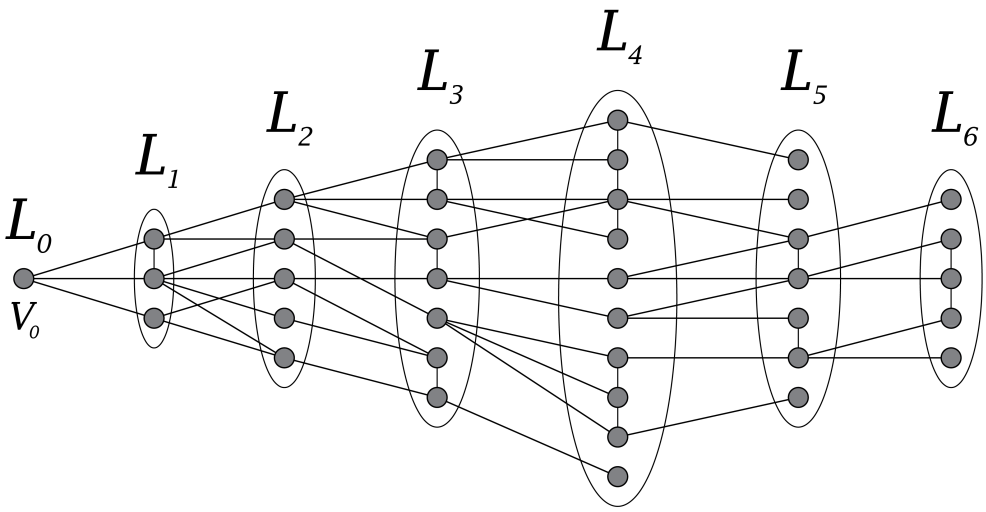

Figure 2.1: An example level structure rooted at v_0

consecutive parts. Figure 2.2 displays an initial partition P_0 with $K = 3$ and delimiters $\{(2,3),(4,5)\}$.

4. *Adjusting the partition P_0 to obtain more balanced parts:* If the balance of the initial partition P_0 is found to be unsatisfactory, they utilize the first two steps of Dulmage-Mendelsohn decomposition algorithm [12] to obtain a more balanced partition P_1 through exchanging vertices between consecutive parts.

5. *Finding a vertex separator between each two consecutive parts:* For each two consecutive parts $\mathcal{V}_i$ and $\mathcal{V}_{i+1}$, a bipartite graph of the boundary vertices and the separating edges is constructed and the minimum vertex cover of this bipartite graph constitute the vertex separator $\mathcal{S}_i$. This results in the partition $P_2 = \{\mathcal{W}_1, \mathcal{S}_1, \mathcal{W}_2, \mathcal{S}_2, \mathcal{W}_3, \ldots, \mathcal{S}_{K-1}, \mathcal{W}_K\}$ where the vertices of separators $\mathcal{S}_i$'s are removed from the parts $\mathcal{V}_i$'s forming $\mathcal{W}_i$'s, i.e., $\mathcal{W}_i = \mathcal{V}_i - (\mathcal{S}_{i-1} \cup \mathcal{S}_i)$. In P_2, part $\mathcal{W}_i$ is only adjacent to its left separator $\mathcal{S}_{i-1}$ and its right separator $\mathcal{S}_i$, whereas separator $\mathcal{S}_i$ is only adjacent to its left part $\mathcal{W}_i$, its right part $\mathcal{W}_{i+1}$, its left separator $\mathcal{S}_{i-1}$ and its right separator $\mathcal{S}_{i+1}$ (see Figure 4.5 for an example of this structure where parts are labeled with $\mathcal{V}_i$ instead of $\mathcal{W}_i$). Note that no two consecutive parts $\mathcal{W}_i$ and $\mathcal{W}_{i+1}$'s are adjacent anymore.

6. *Refining vertex separators:* Finally, an iterative refinement process on P_2 is

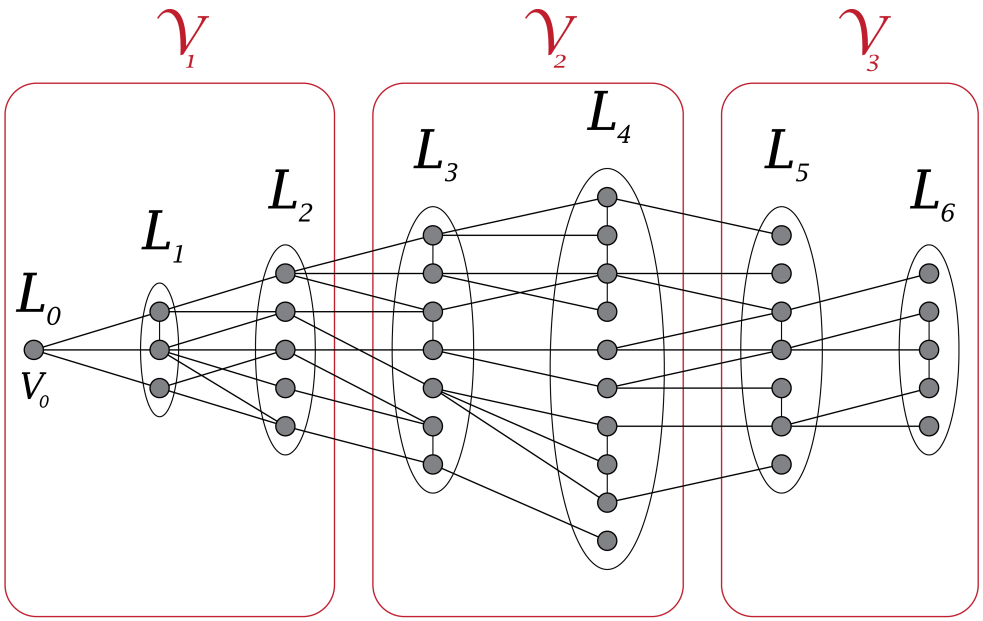

Figure 2.2: An example initial partition P_0 of level structure given in Figure 2.1

used to decrease the size of the separators by utilizing the node separator refinement algorithm of [22]. At each iteration of this algorithm, the first two steps of Dulmage-Mendelsohn decomposition algorithm is used in order to find the set of vertices $Y \subset \mathcal{S}_i$ in separator $\mathcal{S}_i$ whose adjacency set $Adj(Y, (\mathcal{W}_i \cup \mathcal{W}_{i+1}))$ in its left or right part is smaller than itself, i.e., $|Adj(Y, (\mathcal{W}_i \cup \mathcal{W}_{i+1}))| < |Y|$. Then $Adj(Y, (\mathcal{W}_i \cup \mathcal{W}_{i+1}))$ is removed from the corresponding part and replaced in separator $\mathcal{S}_i$ and Y is removed from $\mathcal{S}_i$ and replaced in the corresponding part. Through the iterations, separator $\mathcal{S}_i$'s are selected in the order of their decreasing size and this replacement can be done unless it results an unsatisfactory imbalance on part weights.

Chapter 3

Background

3.1 Standard Graph Model for Representing Sparse Matrices

In the standard graph model, an $N \times N$ square and symmetric matrix $A = (a_{ij})$ is represented as an undirected graph $\mathrm{G}(A) = (\mathcal{V}, \mathcal{E})$ with N vertices. Vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ respectively represent the rows/columns and off-diagonal nonzeros of matrix A. $\mathcal{V}$ contains one vertex v_i for each row/column i. $\mathcal{E}$ contains one edge e_{ij} that connects the vertices v_i and v_j for each symmetric nonzero pair a_{ij} and a_{ji} in A.

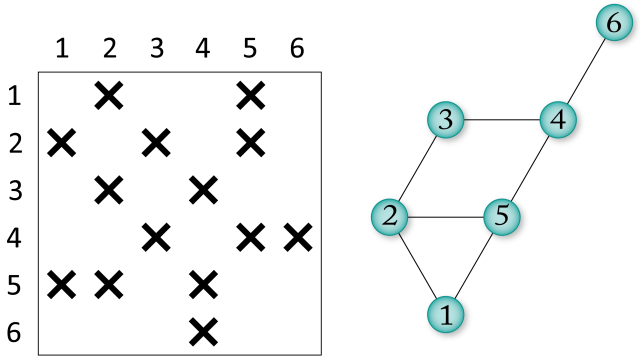

Figure 3.1: A matrix and its standard graph representation

3.2 Graph Partitioning by Vertex Separator (GPVS)

For a given undirected graph $G = (\mathcal{V}, \mathcal{E})$, we use the notation $Adj(v_i)$ to denote the set of vertices that are adjacent to vertex v_i in graph G. That is, $Adj(v_i) = \{v_j : (v_i, v_j) \in \mathcal{E}\}$. We extend this operator to include the adjacency set of a vertex subset $\mathcal{V}' \subseteq \mathcal{V}$, i.e., $Adj(\mathcal{V}') = \bigcup_{v_i \in \mathcal{V}'} Adj(v_i) - \mathcal{V}'$. Two vertex subsets $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{V}'' \subseteq \mathcal{V}$ are said to be adjacent if there exists a pair of vertices $v_i \in \mathcal{V}'$ and $v_j \in \mathcal{V}''$ such that $(v_i, v_j) \in \mathcal{E}$ (i.e., $Adj(\mathcal{V}') \cap \mathcal{V}'' \neq \emptyset$ or equivalently $Adj(\mathcal{V}'') \cap \mathcal{V}' \neq \emptyset$) and non-adjacent otherwise.

A vertex subset $\mathcal{S}$ is a K-way *vertex separator* if the subgraph induced by the vertices in $\mathcal{V} - \mathcal{S}$ has at least K connected components. $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{S}\}$ is a K-way vertex partition of G by vertex separator $\mathcal{S} \subseteq \mathcal{V}$ if all parts are nonempty (i.e., $\mathcal{V}_k \neq \emptyset$ for $k = 1, \ldots, K$), all parts and the separator are pairwise disjoint (i.e., $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ and $V_i \cap \mathcal{S} = \emptyset$ for $i, j = 1, 2, \ldots, K$ and $i \neq j$), the union of the parts and the separator gives $\mathcal{V}$ (i.e., $\bigcup_{i=1}^{K} \mathcal{V}_i \cup \mathcal{S}$), and the vertex parts are pairwise nonadjacent (i.e., $Adj(\mathcal{V}_k) \subseteq \mathcal{S}$ for $k = 1, \ldots, K$). $\mathcal{V}_k \cap Adj(\mathcal{S})$ is said to be the boundary vertex set of part $\mathcal{V}_k$.

Figure 3.2 shows an example graph and an example vertex separator on the graph.

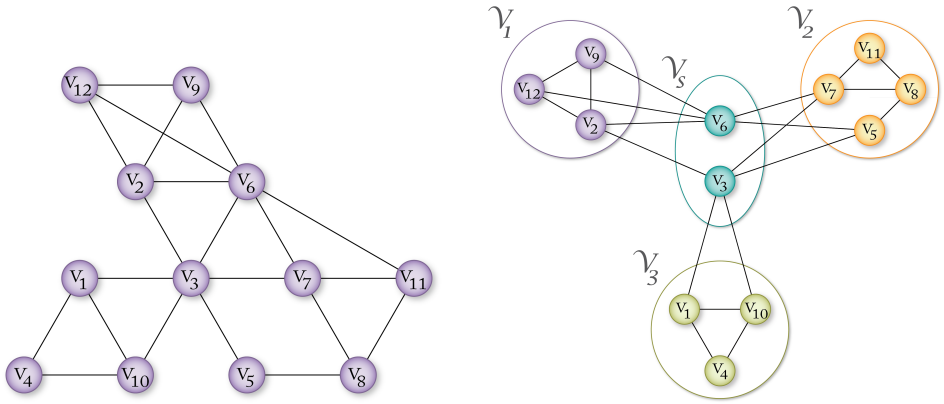

Figure 3.2: An example graph G and an example 3-way separator Π_{VS} of G

In the GPVS problem, the partitioning objective is to minimize the separator size,

which is usually defined as the number of vertices in the separator, i.e.,

$$Separatorsize(\Pi_{VS}) = |\mathcal{S}|. \tag{3.1}$$

The partitioning constraint is to maintain a balance criterion on the part weights, which is usually defined as

$$\max_{1 \leq k \leq K} \{W(\mathcal{V}_k)\} \quad \leq \quad (1+\epsilon)W_{avg}. \tag{3.2}$$

Here, ϵ is the maximum imbalance ratio allowed and $W_{avg} = \sum_{k=1}^{K} W(\mathcal{V}_k)/K$ is the average part weight, where

$$W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i), \tag{3.3}$$

and $w(v_i)$ is the weight associated with vertex v_i.

3.3 Recursive Bipartitioning Paradigm

The RB paradigm has been widely and successively utilized in K-way graph/hypergraph partitioning. In the RB scheme for K-way GPVS, firstly a 2-way GPVS $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{S}\}$ of the original graph $G = G[\mathcal{V}]$ is obtained and then this 2-way Π_{VS} is decoded to construct two subgraphs using the separator-vertex removal scheme to capture the K-way separator size. The separator-vertex removal scheme discards all separator vertices of the 2-way Π_{VS}, since they contribute to the K-way separator size only once, thus inducing vertex induced subgraphs $G[\mathcal{V}_1]$ and $G[\mathcal{V}_2]$. Then 2-way GPVS is recursively applied on both $G[\mathcal{V}_1]$ and $G[\mathcal{V}_2]$. This procedure continues until the desired number of parts is reached in lg_2K recursion levels, assuming K is a power of 2.

In the forthcoming discussions, we utilize the concept of an RB tree which is a full and complete (for K is a power of 2) binary rooted tree. Each node of an RB tree represents a vertex subset of $\mathcal{V}$ as well as the respective induced subgraph on which a 2-way GPVS to be applied. Note that the root node represents both the original vertex set $\mathcal{V}$ and the original graph G.

3.4 Graph/Hypergraph Partitioning with Fixed Vertices

Graph/hypergraph partitioning with fixed vertices has been initially used for RB-based VLSI layout design with terminal propagation [1], and recently used for solving the repartitioning/remapping problem encountered in the parallelization of irregular applications [2, 6, 7].

In graph/hypergraph partitioning with fixed vertices, there exists an additional constraint on the part assignment of some vertices. That is, some vertices, which are referred to as fixed vertices, are pre-assigned to parts prior to the partitioning operation, with the constraint that, at the end of the partitioning, fixed vertices will remain in the part to which they are pre-assigned. We use the notation $\mathcal{F}_k$ to denote the subset of vertices that are fixed to part $\mathcal{V}_k$, for $k = 1, 2, \ldots, K$. The remaining vertices (i.e., vertices in $\mathcal{V} - \bigcup_{k=1}^{K} \mathcal{F}_k$) are referred to as the free vertices since they can be assigned to any part. In GPVS with fixed vertices, free vertices can be assigned to the separator as well as to the parts.

Chapter 4

Ordered GPVS Formulation

In order to formulate the A-to-A_{BDO} transformation problem as a graph theoretical problem, we define a variant of the K-way GPVS problem which is referred to as the *ordered GPVS (oGPVS) problem* .

4.1 Ordered GPVS Problem Definition

In the oGPVS problem, we use a special form of vertex separator which is referred as the *ordered Vertex Separator (oVS)*. In oVS of a given graph G, there exists an order on the vertex parts and the overall separator is partitioned into an ordered set $\mathcal{S} =< \mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{K-1} >$ of mutually disjoint $K-1$ subseparators in such a way that:

(i) Each vertex in subseparator $\mathcal{S}_k$ connects vertices only in successive parts $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$, for $k = 1, 2, ..., K-1$.

(ii) Edges between subseparators are restricted to be between only successive supseperators, i.e., $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$ for $k = 1, 2, ..., K-2$.

Here we refer $\mathcal{S}_k$ as the right subseparator of $\mathcal{V}_k$ and the left subseparator of $\mathcal{V}_{k+1}$. We introduce the following formal definitions for oVS and oGPVS problem:

Definition 1 *Ordered Vertex Separator* Π_{oVS}*:* $\Pi_{oVS} = \{<\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K>; \mathcal{S}\}$ *is a* K*-way ordered vertex partition of* $G = (\mathcal{V}, \mathcal{E})$ *by an ordered vertex seperator* $\mathcal{S} =< \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{K-1} >$ *if each subseparator* $\mathcal{S}_k$ *are nonempty; all parts and subseparators are pairwise disjoint; the union of parts and separators gives* $\mathcal{V}$*; parts are pairwise non-adjacent; only successive subseparators can be pairwise adjacent; successive parts* $\mathcal{V}_k$ *and* $\mathcal{V}_{k+1}$ *are connected by the vertices of the subseparator* $\mathcal{S}_k$ *between these two parts.*

Figure 4.1 displays the general structure of an oVS for parts $\mathcal{V}_{k-1}, \mathcal{V}_k$ and $\mathcal{V}_{k+1}$.

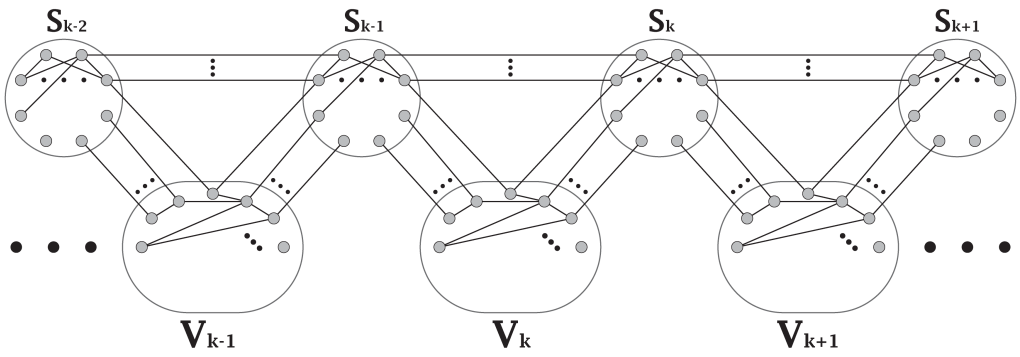

Figure 4.1: General structure of an oVS

Definition 2 *oGPVS Problem: Given a graph* $G = (\mathcal{V}, \mathcal{E})$*, an integer* K*, and a maximum allowable imbalance ratio* ϵ*, the oGPVS problem is finding a* K*-way ordered vertex separator* $\Pi_{oVS}(G) = \{< \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K >; \mathcal{S}\}$ *of* G *by a vertex separator* $\mathcal{S} =< \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{K-1} >$ *that minimizes the overall separator size* $|\mathcal{S}| = \sum_{k=1}^{K-1} |\mathcal{S}_k|$ *while satisfying the balance criterion on the weights of* K *parts given in (3.2).*

4.2 Formulation

The following theorem shows how the A-to-A_{BDO} permutation problem can be formulated as an oGPVS problem.

Theorem 1 *Let $G(A) = (\mathcal{V}, \mathcal{E})$ be the standard graph representation of a given sparse matrix A where weight of each vertex v_i is set to be equal to the number of nonzeros in row/column i. A K-way oVS $\Pi_{oVS} = \{<\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K>; \mathcal{S}\}$ of $G(A)$ can be decoded as a partial permutation of A to a K-way BDO form A_{BDO}, where the vertices of part $\mathcal{V}_k$ and subseparator $\mathcal{S}_k$ constitute the rows/columns of the block $A_{k,k}$ and $C_{k,k}$ respectively. Thus,*

- *minimizing the separator size $|\mathcal{S}| = \sum_{k=1}^{K} |\mathcal{S}_k|$ corresponds to minimizing the sum of the rows/columns of the coupling diagonal blocks*
- *maintaining balance on the part weights relates to maintaining balance on the nonzero counts of the diagonal blocks.*

Proof Consider a K-way oVS $\Pi_{oVS} = \{<\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K>; \mathcal{S}\}$ of $G(A)$. Π_{oVS} can be decoded as a partial permutation on the rows and columns of A to induce a permuted matrix A^{π} as follows: The rows/columns corresponding to the vertices in $\mathcal{V}_k$ are ordered after the rows/columns corresponding to the vertices in $\mathcal{S}_{k-1}$ and before the rows/columns corresponding to the vertices in $\mathcal{S}_k$. In a dual manner, the rows/columns corresponding to the vertices in $\mathcal{S}_k$ are ordered after the rows/columns corresponding to the vertices in $\mathcal{V}_k$ and before the rows/columns corresponding to the vertices in $\mathcal{V}_{k+1}$. Note that Π_{oVS} induces a partial permutation, since the rows/columns corresponding to the vertices in the same part or in the same separator can be ordered arbitrarily. Also note that Π_{oVS} induces a symmetric permutation on the rows and columns of matrix A since each vertex v_i of $G(A)$ represents both row i and column i of A.

In the permuted matrix A^{π}, the vertices of part $\mathcal{V}_k$ constitute the rows/columns of the diagonal subblock $A_{k,k}$ of D_k and the vertices of subseparator $\mathcal{S}_k$ constitutes the rows/columns of the coupling diagonal block $C_{k,k}$ between D_k and D_{k+1}. Since we have $Adj(\mathcal{V}_k) = \mathcal{S}_{k-1} \cup \mathcal{S}_k$ and $Adj(\mathcal{V}_k) \cap Adj(\mathcal{V}_{k+1}) = \mathcal{S}_k$ by the definition of oVS, the overlaps between the diagonal blocks D_k's are restricted to be only between the successive D_k's, and $C_{k,k}$ constitute the overlap between D_k and D_{k+1}. Thus permuted matrix A^{π} is a BDO form of matrix A.

Since the vertices in $\mathcal{S}_k$ constitute the rows/columns of the coupling diagonal block

$C_{k,k}$, minimizing the separator size $|\mathcal{S}|$ corresponds to minimizing the sum of the number of the rows/columns in the coupling diagonal blocks.

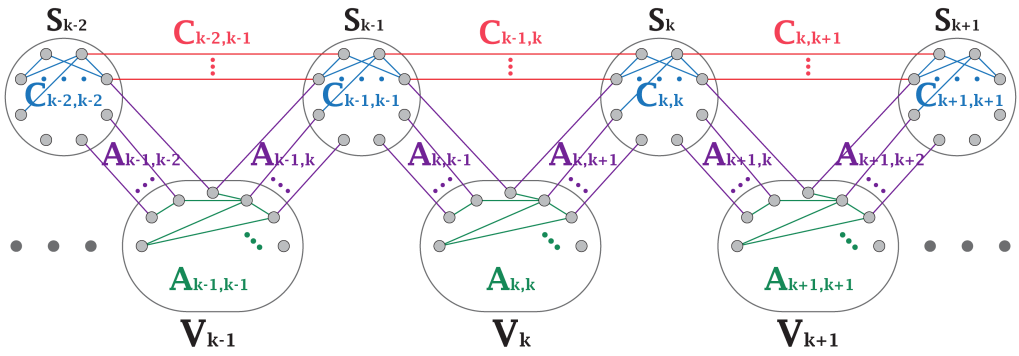

Figure 4.2: Correspondence between the nonzeros of block D_k and the edges of $\mathcal{S}_{k-1} \cup \mathcal{V}_k \cup \mathcal{S}_k$.

Here we show that balancing on the part weights relates to the balancing of the nonzero counts in the diagonal blocks. For this purpose, we mention the association between the edges of $G(A)$ in oVS form and the nonzeros of $A^{\pi} = A_{BDO}$ induced by Π_{oVS}. We introduce Figure 4.2 in order to clarify the forthcoming discussion. The nonzeros in the diagonal subblocks $A_{k,k}$ and $C_{k,k}$ of B_k respectively correspond to the internal edges of part $\mathcal{V}_k$ and subseparator $\mathcal{S}_k$. The nonzeros in the off-diagonal subblocks $A_{k,k+1}$ and $A^T_{k,k+1}$ of D_k correspond to the edges connecting the vertices in $\mathcal{V}_k$ and $\mathcal{S}_k$. The nonzeros in the off-diagonal subblocks $C_{k-1,k}$ and $C^T_{k-1,k}$ of D_k correspond to the edges connecting the vertices in successive subseparators $\mathcal{S}_{k-1}$ and $\mathcal{S}_k$. Thus, the weight of a part $\mathcal{V}_k$ computed according to (3.3) gives $W(\mathcal{V}_k) = nnz(A_{k,k-1}) + nnz(A_{k,k}) + nnz(A_{k,k+1})$, where $nnz(\cdot)$ denotes the number of nonzeros in the respective matrix. Since $nnz(A^T_{k,k-1}) = nnz(A_{k,k-1})$ and $nnz(A^T_{k,k+1}) = nnz(A_{k,k+1})$, $W(\mathcal{V}_k)$ represents the sum of the nonzero counts of diagonal block $A_{k,k}$ plus one of the two off-diagonal blocks $A_{k,k-1}$ and $A^T_{k,k-1}$ plus one of the two off-diagonal blocks $A_{k,k+1}$ and $A^T_{k,k+1}$. One possible nonzero-count coverage of $W(\mathcal{V}_k)$ is shown in (4.1) as highlighted submatrices.

$$D_k = \begin{bmatrix} C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\ A^T_{k,k-1} & A_{k,k} & A_{k,k+1} \\ C^T_{k-1,k} & A^T_{k,k+1} & C_{k,k} \end{bmatrix} \tag{4.1}$$

Note that $W(\mathcal{S}_{k-1}) + W(V_k) + W(\mathcal{S}_k)$ computed in the vertex induced subgraph $G[\mathcal{S}_{k-1} \cup \mathcal{V}_k \cup \mathcal{S}_k]$ of $G(A)$ gives $nnz(D_k)$. Thus, $W(\mathcal{V}_k)$ can be considered to approximate $nnz(D_k)$ when the number of vertices and edges of vertex induced subgraph $G[\mathcal{S}_{k-1} \cup \mathcal{S}_k]$ of $G(A)$ are small, which is partially implied by the partitioning objective of minimizing the separator size. □

Figure 4.3 and 4.4 respectively show a sample 24×24 matrix A which contains 116 nonzeros and the standard graph representation G of A which contains 24 vertices and 46 edges. Figure 4.5 shows a 4-way oVS $\Pi_{oVS}(G) = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4; \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$ of G, where $\mathcal{V}_1$,$\mathcal{V}_2$,$\mathcal{V}_3$ and $\mathcal{V}_4$ respectively contain 4, 5, 4 and 4 vertices, and $\mathcal{S}_1$,$\mathcal{S}_2$ and $\mathcal{S}_3$ respectively contain 2, 3 and 2 vertices. Figure 4.6 shows a BDO form of the sample matrix A given in Figure 4.3, which is induced by $\Pi_{oVS}(G)$ given in Figure 4.5. As seen in Figure 4.6, the BDO form respectively contains diagonal blocks D_1, D_2, D_3 and D_4 of dimensions 6×6, 10×10, 9×9 and 6×6, and overlapping blocks $C_{1,1}$, $C_{2,2}$ and $C_{3,3}$ of dimensions 2×2, 3×3, and 2×2 between diagonal blocks D_1 and D_2, D_2 and D_3, and D_3 and D_4.

4.3 Parallel Application Requirements

Here we will briefly examine the communication and computation requirements of the parallel implementation of an explicit formulation of the multiplicative schwarz preconditioner given in [18] in order to show the correspondence between its efficient parallelization and the constraint and objective of the proposed oGPVS formulation. In this parallel implementation, each processor k stores diagonal block D_k and its LU factors as well as the kth overlapping subvectors of all column vectors involved in the iterative solution of $A^\pi x^\pi = b^\pi$, where $x^\pi = P^T x$ and $b^\pi = Pb$. For the simplicity of the forthcoming discussion, we will omit the "$^\pi$" superscripts which denote the permuted matrix and vectors. For example, x_k denotes the subvector of x that corresponds to the columns of D_k, where x_k is partitioned into three subsubvectors x_k^1, x_k^2 and x_k^3 that respectively correspond to the columns of $C_{k-1,k-1}$, $A_{k,k}$ and $C_{k,k}$. So x_k overlaps with x_{k-1} through x_{k-1}^3 and x_k^1, and overlaps with x_{k+1} through x_k^3 and x_{k+1}^1. Each iteration involves a residual computation step and a preconditioning step

[18].

The residual computation step involves a local sparse matrix-vector multiply (SpMxV) operation of the form $z_k = \hat{D}_k x_k$ for updating the local residual vector through the local linear vector operation $r_k = b_k - z_k$, in each processor k. Here $\hat{D}_k$ is the diagonal block D_k from which the coupling diagonal subblock $C_{k,k}$ is zeroed as shown below:

$$\hat{D}_k = \begin{bmatrix} C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\ A_{k,k-1}^T & A_{k,k} & A_{k,k+1} \\ C_{k-1,k}^T & A_{k,k+1}^T & 0 \end{bmatrix} \tag{4.2}$$

The preconditioning step involves the solution of a local linear system of the form $D_k y_k = r_k$ for the update of the local solution vector through the linear vector operation $x_k = x_k + y_k$ in each processor k. y_k is obtained through performing local forward and backward substitution operations on the LU factors of D_k. The local LU factorizations of D_k matrices are performed in a parallel pre-processing step [18]. The preconditioning step also involves a SpMxV operation of the form $y_k^3 = C_{k,k} y_k^3$, where y_k^3 is the subvector of y_k that corresponds to the rows of $C_{k,k}$. So maintaining balance on the part weights relates to maintaining balance on the computational loads of processors during the iterations.

In each residual computation step, processor k sends z_k^1 to processor $k-1$, and sends z_k^3 to processor $k+1$. In each preconditioning step, processor k sends y_k^1 to processor $k-1$, and sends y_k^3 to processor $k+1$. Hence, the partitioning objective of minimizing the overall separator size corresponds to minimizing the total communication volume. Furthermore, as mentioned in [19], minimizing the overall separator size corresponds to minimizing the upper bound on the convergence rate of the iterative method.

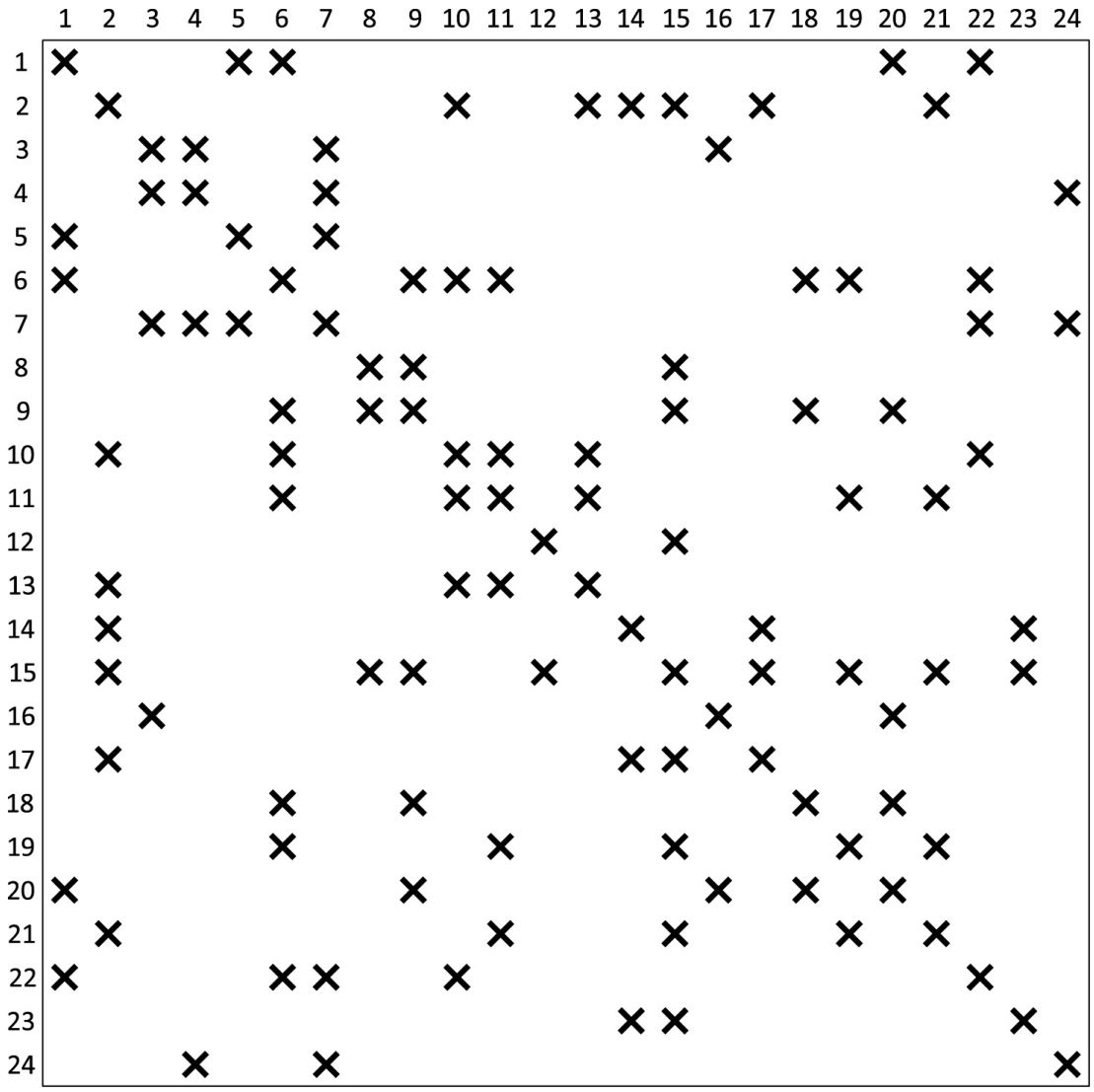

Figure 4.3: Sample matrix A

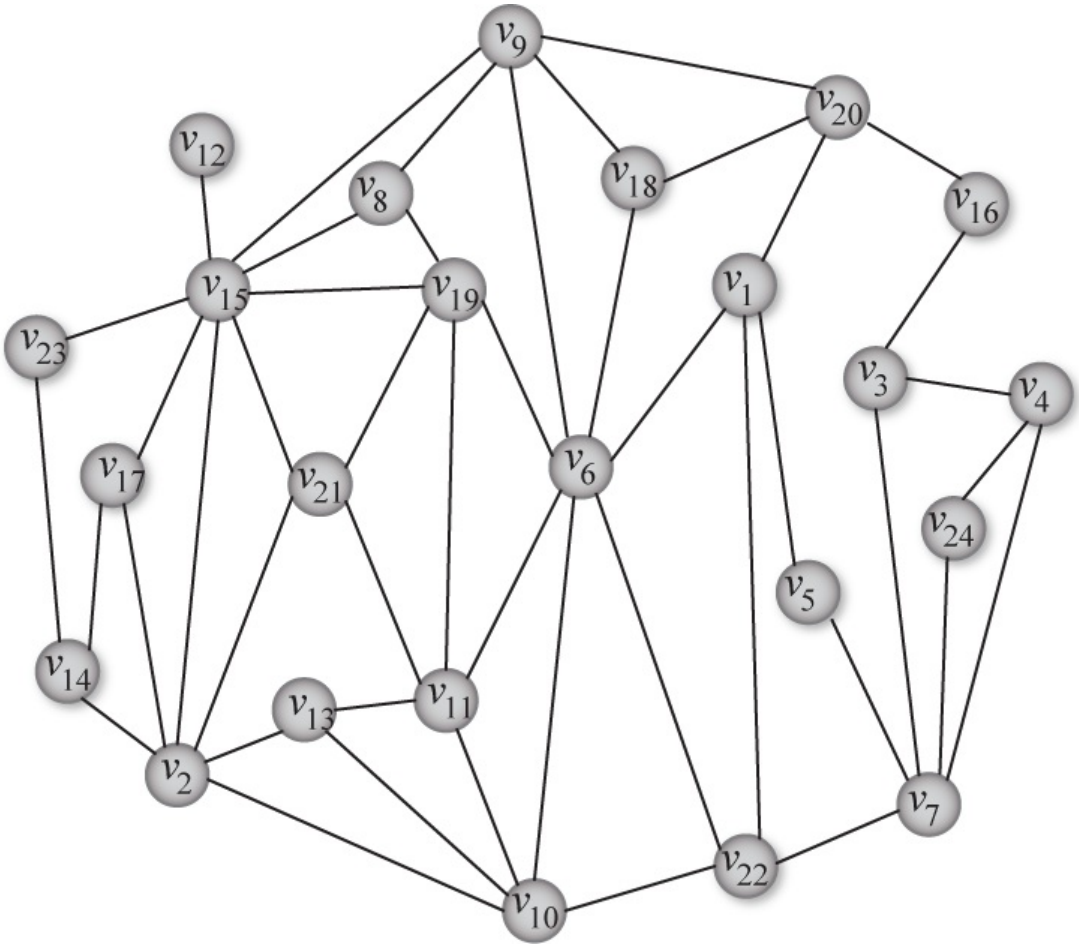

Figure 4.4: Standard graph representation $G(A)$ of A given in Figure 4.3

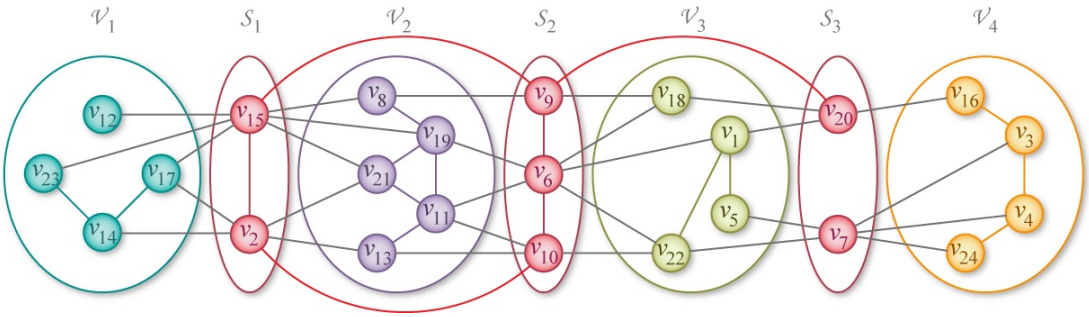

Figure 4.5: A 4-way oVS form of $G(A)$ given in Figure 4.4

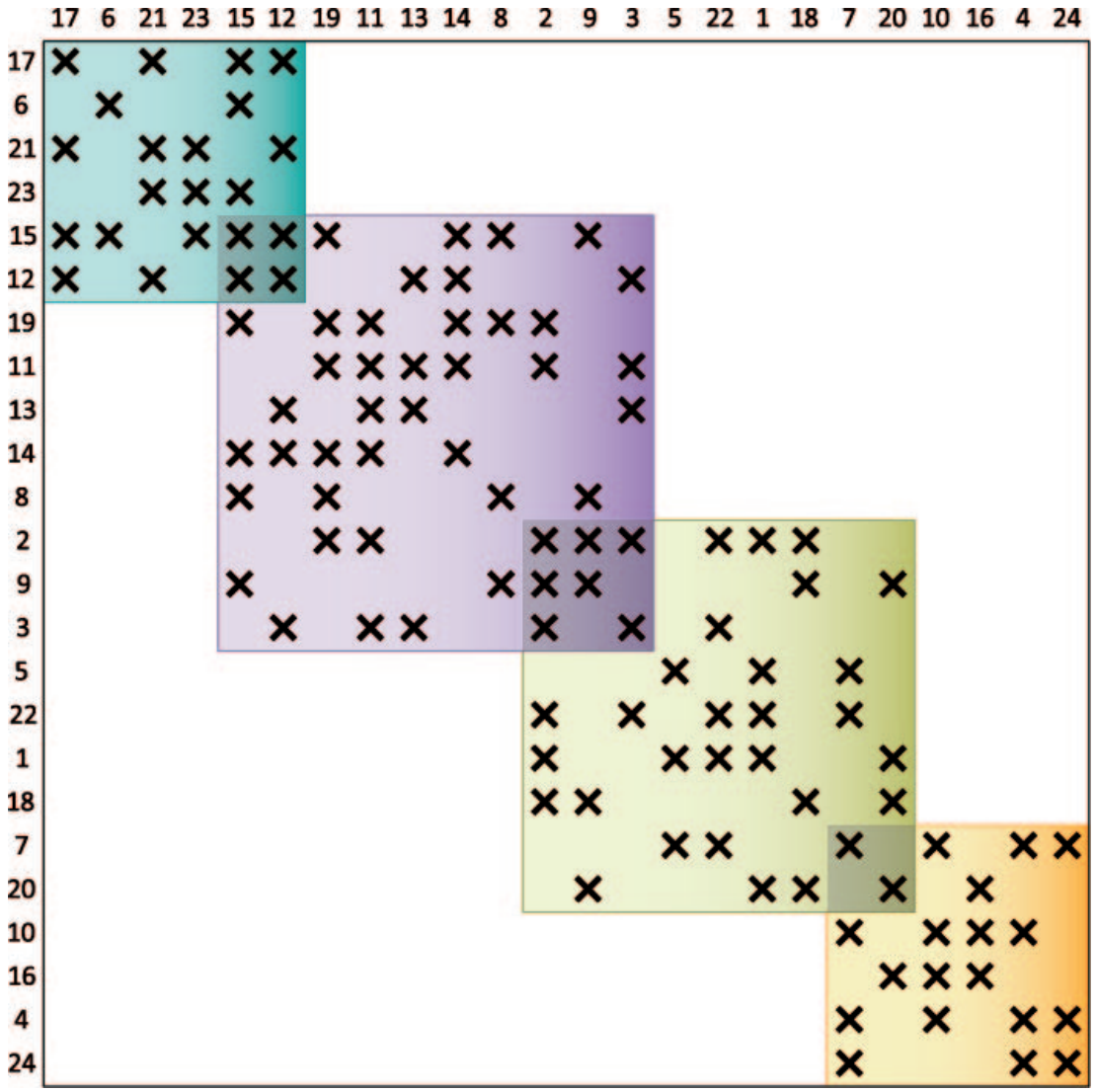

Figure 4.6: BDO form of A permuted by 4-way oVS of $G(A)$ given in Figure 4.5

Chapter 5

Recursive Graph Bipartitioning Model with Fixed Vertices

In this section, we show how we solve the oGPVS problem by utilizing 2-way GPVS problem with fixed vertices within the RB paradigm.

5.1 Theoretical Foundations

The following theorem and corollary lays down the basis for our formulation to obtain a K-way oVS of a given graph $G = (\mathcal{V}, \mathcal{E})$.

Theorem 2 *For any disjoint vertex subset pair $\mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$, G has a K-way oVS $\Pi_{oVS} = \{< \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K >; \mathcal{S}\}$ such that $\mathcal{B}_L \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ and $\mathcal{B}_R \subseteq \mathcal{S}_{K-1} \cup \mathcal{V}_K$ if and only if the distance between any two vertices $v_i \in \mathcal{B}_L$ and $v_j \in \mathcal{B}_R$ is at least $K-2$.*

Proof *(If)* Consider the level structure initiated with $\mathcal{B}_L$, i.e., $\mathcal{L}_0 = \mathcal{B}_L$. Since the distance between any vertices $v_i \in \mathcal{B}_L$ and $v_j \in \mathcal{B}_R$ is at least $K-2$, $v_j \in \mathcal{L}_\ell$ s.t. $\ell \geq K-2$, for any $v_j \in \mathcal{B}_R$. We can construct a K-way oVS Π_{oVS} such as $\mathcal{S}_k = \mathcal{L}_{k-1}$ for $1 \leq k < K-1$ and $\mathcal{S}_{K-1} = \bigcup_{k \geq K-1} \mathcal{L}_{k-1}$. Since $\mathcal{B}_L = \mathcal{S}_1$, $\mathcal{B}_L \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$. Due to the construction, $\mathcal{B}_R \subseteq \mathcal{V}_K \cup \mathcal{S}_{K-1}$ since $v_j \in \mathcal{S}_{K-1}$ for any $v_j \in \mathcal{B}_R$.

(Only If) Consider a K-way oVS such that $\mathcal{B}_L \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ and $\mathcal{B}_R \subseteq \mathcal{V}_K \cup \mathcal{S}_{K-1}$. Consider any vertex pair $v_i \in \mathcal{B}_L$ and $v_j \in \mathcal{B}_R$. It is clear that, the minimum distance between v_i and v_j occurs when $v_i \in \mathcal{S}_1$ and $v_j \in \mathcal{S}_{K-1}$. Due to the oVS structure, any path between a vertex of $\mathcal{S}_1$ and a vertex of $\mathcal{S}_{K-1}$ contains at least $K-2$ intermediate vertices one from each subseparator $\mathcal{S}_k$ (for $k = 2, 3, \ldots, K-2$). So, the minimum distance between v_i and v_j is at least $K-1$. □

Corollary 1 *A graph G has a K-way oVS if and only if the diameter of G is at least $K-2$.*

Proof G has diameter of size at least $K-2$ if and only if there exists two vertices v_i and v_j such that $\delta(v_i, v_j) \geq K-2$. Having such two vertices implies the existance of a K-way oVS of G such that $v_i \in \mathcal{V}_1 \cup \mathcal{S}_1$ and $v_j \in \mathcal{S}_{K-1} \cup \mathcal{V}_K$ due to Theorem 2. On the other hand, by definition, if G has a K-way oVS then there exists two vertices $v_i \in \mathcal{S}_1$ and $v_j \in \mathcal{S}_{K-1}$. Then, Theorem 2 implies that $\delta(v_i, v_j) \geq K-2$. □

5.2 Recursive oGPVS Algorithm

Theorem 2 and Corollary 1 give the necessary and sufficient conditions for finding a K-way oVS of a given graph $G = (\mathcal{V}, \mathcal{E})$. However, a new scheme is needed to be applied during each RB step to satisfy the feasibility condition for the resulting K-way GPVS to be a K-way oVS. For this purpose, we propose a left-to-right bipartitioning approach together with a novel vertex fixation scheme so that a GPVS tool that supports partitioning with fixed vertices can be effectively and efficiently utilized. Algorithm 1 shows the initial invocation of the recursive oGPVS algorithm, where Algorithm 2 displays the basic steps of the proposed RB-based oGPVS algorithm that utilizes the proposed vertex fixation scheme.

As seen in Algorithm 1, for the first RB step of recursive oGPVS algorithm, $\mathcal{B}_L$ consists of a single pseudo-peripheral vertex v_L which is found by using the pseudo-peripheral node finder algorithm given in [15]. One of the vertices that has a maximum

Algorithm 1 Initialization

Require: Graph $G = (\mathcal{V}, \mathcal{E})$, integer K
1: Find a pseudo-peripheral vertexv_L
2: Find a furthest vertex v_R to v_L using BFS
3: **if** distance between v_L and v_R is less than $K-2$ **then**
4: **return** "G is not partitionable into K-way oVS"
5: **else**
6: $\mathcal{B}_L \leftarrow \{v_L\}$
7: $\mathcal{B}_R \leftarrow \{v_R\}$
8: $\Pi_{oVS} \leftarrow$oGPVS$(G, \mathcal{B}_L, \mathcal{B}_R, K)$
9: **return** Π_{oVS}

distance to the selected pseudo-peripheral vertex is taken as the single vertex v_R constituting $\mathcal{B}_R$. According to Theorem 2, the oGPVS algorithm can be terminated at this initial stage if the shortest path distance between v_L and v_R is less than $K-2$.

Algorithm 2 displays the oGPVS function whose inputs are a graph G, left and right boundary vertex sets $\mathcal{B}_L$ and $\mathcal{B}_R$ of G, and an integer K which is the number of parts that G is to be partitioned into. After the execution of this function, a K-way oVS of the graph G is returned. Note that G and K are the current inputs of the oGPVS function although they also denote the initial graph and integer. As will become clear later, left and right boundary vertex sets $\mathcal{B}_L$ and $\mathcal{B}_R$ are needed to gather the information of which vertices are to be fixed to the left and right parts while applying vertex fixation scheme.

As seen in line 1 of Algorithm 2, the oGPVS function first checks whether the current bipartitioning is an intermediate or final level bipartitioning in the RB tree. Note that $K > 2$ for intermediate level bipartitionings, whereas $K = 2$ for final level bipartitionings. As seen in line 3 of Algorithm 2, at the beginning of each intermediate RB step, the oGPVS function applies the proposed vertex fixation scheme by invoking the FIX-INT-LEVEL function on the current graph G with $\mathcal{B}_L$ and $\mathcal{B}_R$ to obtain the left and right fixed-vertex sets $\mathcal{F}_L$ and $\mathcal{F}_R$. Then in line 4, a 2-way GPVS is invoked on $(G, \{\mathcal{F}_L, \mathcal{F}_R\})$ to obtain $\Pi_{VS}(G) = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$, where $\mathcal{V}_L$ and $\mathcal{V}_R$ are used to denote the left and right parts. In lines 5 and 6, we construct left and right vertex-induced subgraphs $G_L = G[\mathcal{V}_L]$ and $G_R = G[\mathcal{V}_R]$ on which further recursive bipartitioning steps will be applied, since this bipartitioning belongs to an intermediate level of the

Algorithm 2 oGPVS $(G, \mathcal{B}_L, \mathcal{B}_R, K)$

Require: Graph $G = (\mathcal{V}, \mathcal{E})$, boundary vertex sets $\mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$, integer K

if $K > 2$ **then**
  $K' \leftarrow K/2$
  $(\mathcal{F}_L, \mathcal{F}_R) \leftarrow$FIX-INT-LEVEL$(G, \mathcal{B}_L, \mathcal{B}_R, K')$
  $\Pi_{VS} \leftarrow$GPVS$(G, \{\mathcal{F}_L, \mathcal{F}_R\}, 2)$ ▷ $\Pi_{VS} = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$
  $G_L \leftarrow G[\mathcal{V}_L]$
  $G_R \leftarrow G[\mathcal{V}_R]$
  $\mathcal{B}_{LL} \leftarrow \mathcal{B}_L$
  $\mathcal{B}_{LR} \leftarrow Adj(\mathcal{S}) \cap \mathcal{V}_L$
  $\mathcal{B}_{RL} \leftarrow Adj(\mathcal{S}) \cap \mathcal{V}_R$
  $\mathcal{B}_{RR} \leftarrow \mathcal{B}_R$
  $\Pi^L_{oVS} \leftarrow$oGPVS $(G_L, \mathcal{B}_{LL}, \mathcal{B}_{LR}, K')$ ▷ $\Pi^L_{oVS} = \{< \mathcal{V}^L >:< \mathcal{S}^L >\}$
  $\Pi^R_{oVS} \leftarrow$oGPVS $(G_R, \mathcal{B}_{RL}, \mathcal{B}_{RR}, K')$ ▷ $\Pi^R_{oVS} = \{< \mathcal{V}^R >:< \mathcal{S}^R >\}$
  $\Pi_{oVS} \leftarrow \{< \mathcal{V}^L, \mathcal{V}^R >:< \mathcal{S}^L, \mathcal{S}, \mathcal{S}^R >\}$
else
  $(G', \{v_L\}, \{v_R\}) \leftarrow$FIX-FINAL-LEVEL$(G, \mathcal{B}_L, \mathcal{B}_R)$
  $\Pi_{VS} \leftarrow$GPVS$(G', \{\{v_L\}, \{v_R\}\}, 2)$ ▷ $\Pi_{VS} = \{\mathcal{V}'_L, \mathcal{V}'_R; \mathcal{S}\}$
  $\mathcal{V}_L \leftarrow \mathcal{V}'_L - \{v_L\}$
  $\mathcal{V}_R \leftarrow \mathcal{V}'_R - \{v_R\}$
  $\Pi_{oVS} \leftarrow \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$
return Π_{oVS}

RB tree. Note that in order to construct G_L and G_R, we effectively apply the vertex removal scheme on the vertices of subseparator $\mathcal{S}$. That is, each subseparator vertex $v_s \in \mathcal{S}$ is removed during forming G_L and G_R.

In lines 7–10 of Algorithm 2, we determine left and right boundary vertices of both left and right subgraphs G_L and G_R. G_L and G_R respectively inherit their left and right boundary vertex sets from the left and right boundary vertex sets of the parent graph G. That is, the left boundary vertex set $\mathcal{B}_L$ of the current graph G becomes the left boundary vertex set $\mathcal{B}_{LL}$ of G_L, whereas the right boundary vertex set $\mathcal{B}_R$ of G becomes the right boundary vertex set $\mathcal{B}_{RR}$ of G_R. The boundary vertex sets $\mathcal{B}_{LR}$ and $\mathcal{B}_{RL}$ that are formed by the subseparator $\mathcal{S}$ of $\Pi_{VS}(G)$ respectively constitute the right and left boundary vertex sets of G_L and G_R. That is, $Adj(\mathcal{S}) \cap \mathcal{V}_L$ constitutes the right boundary vertex set $\mathcal{B}_{LR}$ of G_L, whereas $Adj(\mathcal{S}) \cap \mathcal{V}_R$ constitutes the left boundary vertex set $\mathcal{B}_{RL}$ of G_R. We should note here that $\mathcal{S}$ will be the right subseparator of the rightmost vertex part and left subseparator of the leftmost vertex part obtained from RB trees rooted at G_L and G_R, respectively.

In lines 11 and 12 of Algorithm 2, we recursively invoke the oGPVS function on the left and right subgraphs G_L and G_R to respectively obtain Π^L_{oVS} and Π^R_{oVS}. Here $\Pi^L_{oVS} = \{<\mathcal{V}^L>:<\mathcal{S}^L>\}$ denotes the resulting $K/2$-way oVS of the left subgraph G_L, where $<\mathcal{V}^L>$ and $<\mathcal{S}^L>$ denote the ordered $K/2$ vertex parts and $K/2-1$ subseparators. Similarly, $\Pi^R_{oVS} = \{<\mathcal{V}^R>:<\mathcal{S}^R>\}$ denotes the resulting $K/2$-way oVS of the right subgraph G_R, where $<\mathcal{V}^R>$ and $<\mathcal{S}^R>$ respectively denote the ordered $K/2$ vertex parts and $K/2-1$ subseparators. Line 13 forms a K-way oVS of G by combining Π^L_{oVS} and Π^R_{oVS} together with the current level subseparator $\mathcal{S}$ as $\Pi_{oVS} = \{<\mathcal{V}^L, \mathcal{V}^R> : <\mathcal{S}^L, \mathcal{S}, \mathcal{S}^R>\}$.

For the final level bipartitionings (lines 15–19 in Algorithm 2), the oGPVS function applies the proposed vertex fixation scheme by invoking the FIX-FINAL-LEVEL function (in line 15) on the current graph G with $\mathcal{B}_L$ and $\mathcal{B}_R$ to obtain augmented graph G'. As will become clear later in Algorithm 4, G' is produced by adding two vertices v_L and v_R, which are respectively fixed to the left and right parts, and a number of associated edges to the current graph G. Then in line 16, a 2-way GPVS is

invoked on $(G', \{\{v_L\}, \{v_R\}\})$ to obtain $\Pi_{VS}(G') = \{\mathcal{V}'_L, \mathcal{V}'_R; \mathcal{S}\}$. Lines 17–18 exclude v_L and v_R from the left and right vertex parts, respectively, to obtain the 2-way oVS in line 19.

Figure 5.1 displays a diagram of three levels of RB process applied on a graph G with left and right boundary vertex sets $\mathcal{B}_L$ and $\mathcal{B}_R$. Solid directed edges connecting graphs to their subgraphs correspond to the edges of the RB tree, whereas the dashed directed edges correspond to the final level bipartitionings. Note that $\mathcal{B}_L$ and $\mathcal{B}_R$ respectively determine the left and right boundary vertex sets of the leftmost and rightmost graphs at each level of the RB tree rooted at G. That is, $\mathcal{B}_L = \mathcal{B}_{LL} = \mathcal{B}_{LLL}$ is the left boundary vertex set of graphs G, G_L and G_{LL}, whereas $\mathcal{B}_R = \mathcal{B}_{RR} = \mathcal{B}_{RRR}$ is the right boundary vertex set of graphs G, G_R and G_{RR}. The internal boundary vertex sets of the RB tree rooted at G are determined by the separators obtained, for example $\mathcal{B}_{LRR} = \mathcal{B}_{LR} = Adj(\mathcal{S}) \cap \mathcal{V}_L$ and $\mathcal{B}_{RLL} = \mathcal{B}_{RL} = Adj(\mathcal{S}) \cap \mathcal{V}_R$. The last level of Figure 5.1 shows the final 2-way GPVS operations performed on the subgraphs of the last level of the RB tree to obtain an 8-way oVS of the initial graph G.

Algorithm 3 FIX-INT-LEVEL $(G, \mathcal{B}_L, \mathcal{B}_R, K')$

Require: Graph $G = (\mathcal{V}, \mathcal{E}), \mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$, integer K'
1: $K' \leftarrow K' - 1$
2: $\mathcal{F}_L \leftarrow$FIX-VERTICES$(G, \mathcal{B}_L, K')$ ▷ fixation to the left part
3: $\mathcal{F}_R \leftarrow$FIX-VERTICES$(G, \mathcal{B}_R, K')$ ▷ fixation to the right part
4: **return** $(\mathcal{F}_L, \mathcal{F}_R)$

Algorithm 4 FIX-FINAL-LEVEL,$(G, \mathcal{B}_L, \mathcal{B}_R)$

Require: Graph $G = (\mathcal{V}, \mathcal{E}), \mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$
1: $\mathcal{V}' \leftarrow \mathcal{V} \cup \{v_\ell\} \cup \{v_r\}$
2: $\mathcal{E}' \leftarrow \mathcal{E} \cup \{(v_\ell, v) : v \in \mathcal{B}_L\} \cup \{(v, v_r) : v \in \mathcal{B}_R\}$
3: $w(v_\ell) \leftarrow w(v_r) \leftarrow 0$
4: $G' = (\mathcal{V}', \mathcal{E}')$
5: **return** $(G', \{v_L\}, \{v_R\})$

As seen in Algorithm 2, we apply two different types of fixation schemes FIX-INT-LEVEL and FIX-FINAL-LEVEL for the intermediate level and final level bipartitionings, respectively. Here, an intermediate level bipartitioning refers to a 2-way GPVS to be applied on a graph at an internal node of the RB tree, whereas a final level bipartitioning refers to a 2-way GPVS to be applied on a graph at a leaf node.

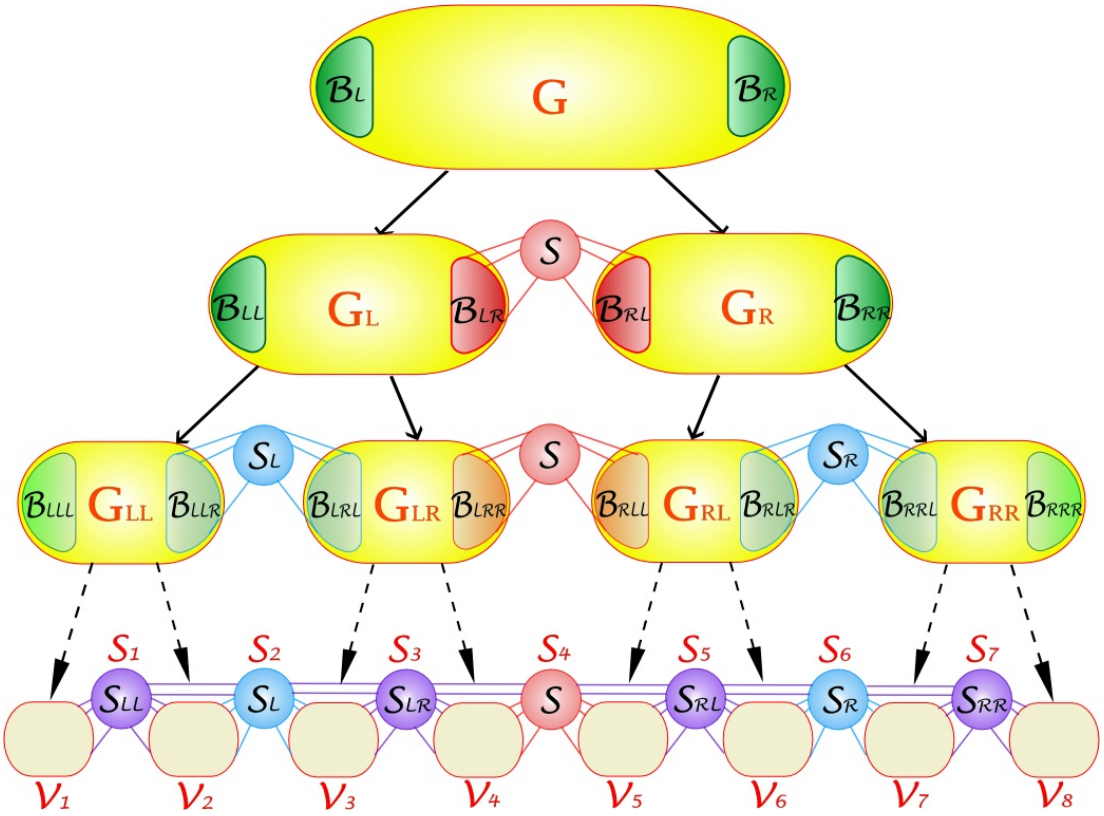

Figure 5.1: A three level RB tree for producing an 8-way oVS of an initial graph G

The FIX-INT-LEVEL function invokes the FIX-VERTICES function twice with K' being equal to $K/2-1$, where K is the input of the current oGPVS function. Here, K' denotes the number of vertex levels to be fixed from the left and right boundary vertex sets–including the boundary vertex sets–of the current graph G. As seen in Algorithm 5, the FIX-VERTICES function utilizes a BFS-like algorithm to identify the vertices whose shortest path distances to a given vertex subset $\mathcal{B}$ are strictly less than a given K' value. The shortest path distance of a vertex v to a vertex subset $\mathcal{U}$ is defined as $\delta(v,\mathcal{U}) = \min_{u\in\mathcal{U}}\{\delta(u,v)\}$, where $\delta(u,v)$ denotes the shortest path distance between two vertices u and v. In the first invocation of the FIX-VERTICES function, vertices whose shortest path distances to $\mathcal{B}_L$ are strictly less than K' are fixed to the left part, whereas in the second invocation vertices whose shortest distances to $\mathcal{B}_R$ are strictly less than K' are fixed to the right part. That is, $\mathcal{F}_L = \{u : \delta(u,\mathcal{B}_L) < K'\}$ and $\mathcal{F}_R = \{u : \delta(u,\mathcal{B}_R) < K'\}$.

For the final level bipartitionings, the FIX-FINAL-LEVEL function augments

Algorithm 5 FIX-VERTICES $(G, \mathcal{B}, K')$

Require: Graph $G = (\mathcal{V}, \mathcal{E}), \mathcal{B} \subseteq \mathcal{V}$, integer K'
1: $\mathcal{F} \leftarrow \emptyset$
2: **for** each vertex $u \in B$ **do**
3: $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{u\}$
4: $\quad d[u] \leftarrow 1$
5: $Q \leftarrow \mathcal{B}$
6: **while** $Q \neq \emptyset$ **do**
7: $\quad u \leftarrow$ DEQUEUE(Q)
8: $\quad$ **for** each vertex $v \in Adj(u)$ **do**
9: $\quad\quad$ **if** $v \notin \mathcal{F}$ **then**
10: $\quad\quad\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{v\}$
11: $\quad\quad\quad d[v] \leftarrow d[u] + 1$
12: $\quad\quad\quad$ **if** $d[v] < K'$ **then**
13: $\quad\quad\quad\quad$ ENQUEUE(Q, v)
14: **return** $\mathcal{F}$

graph G with two zero-weight vertices v_ℓ having $Adj(v_\ell) = \mathcal{B}_L$ and v_r having $Adj(v_r) = \mathcal{B}_R$, and fixes them to the left and right parts, respectively. This vertex fixation scheme introduces the flexibility of assigning the vertices of $\mathcal{B}_L$ and $\mathcal{B}_R$ to the separator.

5.3 A Discussion on the Correctness of oGPVS Algorithm

The left-to-right bipartitioning approach together with the proposed vertex fixation scheme adopted in the recursive oGPVS algorithm given in Algorithm 2 induces a natural ordering on both vertex parts and separators of a graph G in such a way that the final partition is a K-way oVS of G. We should also note that this scheme also induces a restricted 2^ℓ-way oVS at the ℓ^{th} level of the RB tree, for $\ell = 0, 1, \ldots, lg_2K - 1$. Here the restriction refers to the non-adjacency of the consecutive subseparators. As will become clear later, 2-way GPVS operations to be invoked on the leaf level graphs of the RB tree make the consecutive subseparators adjacent in the final K-way oVS.

We provide the following discussion on the correctness of the proposed RB-based

oGPVS algorithm. We include Figure 5.2 for a better understanding of the forthcoming discussion. Without loss of generality, let G be a graph in an intermediate level of the RB tree. Consider a 2-way VS $\Pi_{VS}(G) = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$ of G and let G_L and G_R be the vertex-induced subgraphs by $\mathcal{V}_L$ and $\mathcal{V}_R$, respectively. Let $B_L = Adj(\mathcal{S}) \cap \mathcal{V}_R$ be the left boundary vertex set of G_R and $B_R = Adj(\mathcal{S}) \cap \mathcal{V}_L$ be the right boundary vertex set of G_L. For the sake of correctness of the oGPVS algorithm, the following

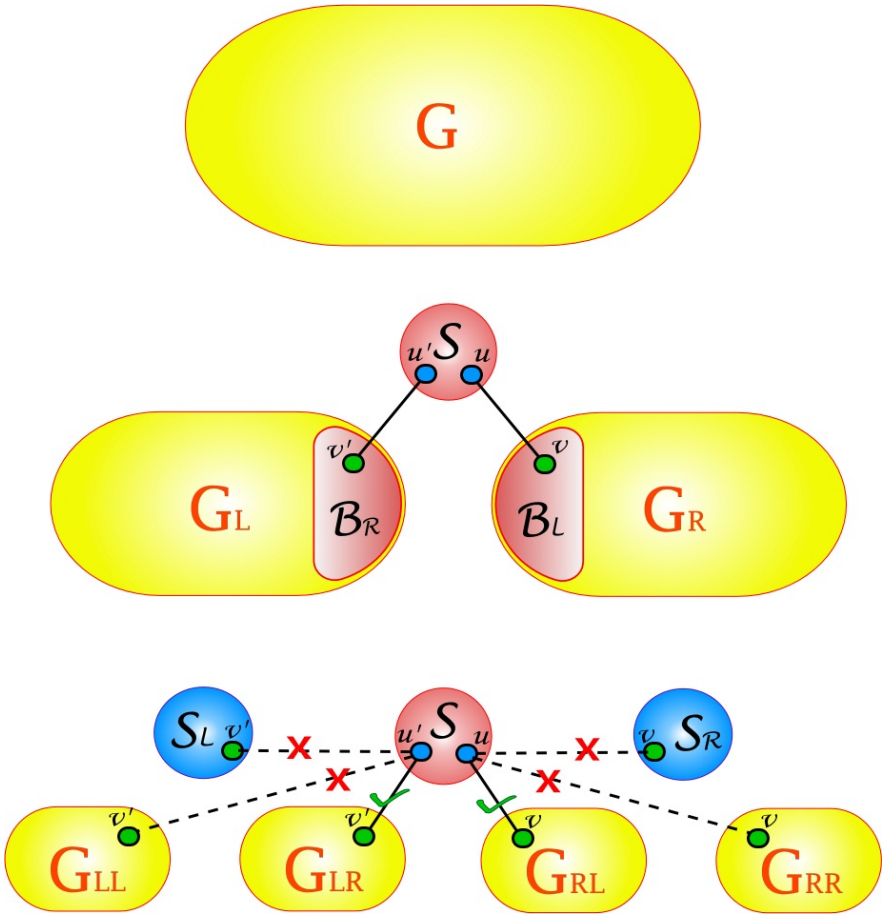

Figure 5.2: Restrictions for boundary vertices

restrictions should be maintained in any 2-way VS $\Pi_{VS}(G_L)$ of G_L and $\Pi_{VS}(G_R)$ of G_R:

(a) If G_L and G_R are intermediate level graphs of the RB tree, the vertices in the left boundary vertex set $\mathcal{B}_L$ of G_R can only be assigned to the left part of $\Pi_{VS}(G_R)$, whereas the vertices in the right boundary vertex set $\mathcal{B}_R$ of G_L can only be assigned to the right part of $\Pi_{VS}(G_L)$.

(b) If G_L and G_R are final level graphs of the RB tree, the vertices in the left boundary vertex set $\mathcal{B}_L$ of G_R can be assigned to the subseparator as well as the left part of $\Pi_{VS}(G_R)$, whereas the vertices in the right boundary vertex set $\mathcal{B}_R$ of G_L can be assigned to the subseparator as well as the right part of $\Pi_{VS}(G_L)$.

We provide the following discussion for the need of restriction (a) on the assignment of the vertices in the left boundary vertex set $\mathcal{B}_L$ of G_R. Consider an edge $(u, v) \in \mathcal{E}(G)$, where $u \in \mathcal{S}$ and $v \in \mathcal{B}_L$ in $\Pi_{VS}(G)$. There are three cases according to the assignment of vertex v in $\Pi_{VS}(G_R) = \{\mathcal{V}_{RL}, \mathcal{V}_{RR}; \mathcal{S}\}$; namely $v \in \mathcal{V}_{RL}$, $v \in \mathcal{V}_{RR}$ and $v \in \mathcal{S}_R$. Case $v \in \mathcal{V}_{RL}$ does not violate the oVS structure at the current level. Case $v \in \mathcal{S}_R$ makes two consecutive subseparators become adjacent in the current level. Although this situation doesn't violate the oVS structure in the current level, it is guaranteed to violate the oVS structure in the subsequent bipartitions of the left and right subgraphs of G_R in the next level since these adjacent subseparators $\mathcal{S}$ and $\mathcal{S}_R$ will not be consecutive anymore in the following levels. Case $v \in \mathcal{V}_{RR}$ immediately violates the oVS structure since edge (u, v) makes separator $\mathcal{S}$ connect two nonconsecutive vertex parts, namely a vertex part in the current level oVS rooted at G_L and the right vertex part of $\Pi_{VS}(G_R)$. A dual discussion holds for the need of restriction (a) on the assignment of the vertices in the right boundary vertex set $\mathcal{B}_R$ of G_L. In Figure 5.2, allowable and disallowable assignments of vertex v are identified by labeling the (u, v) edges with "✓" and "×".

The restriction (b) is a relaxed version of the restriction (a), where the vertices in $\mathcal{B}_L$ and $\mathcal{B}_R$ can also be assigned to the separators of $\Pi_{VS}(G_R)$ and $\Pi_{VS}(G_L)$, respectively. This relaxation is valid, because it has the potential of disturbing the oVS structure only if the left and right subgraphs of $\Pi_{VS}(G_L)$ and $\Pi_{VS}(G)$ are to be further bipartitioned, which is not the case since $\Pi_{VS}(G_L)$ and $\Pi_{VS}(G_R)$ are final level bipartitionings of the RB tree.

It is clear that the fixation scheme given in Algorithms 3 and 4 already achieves fixing the left and right boundary vertex sets in such a way to satisfy the restrictions (a) and (b), respectively. Furthermore, at an intermediate level of RB tree, Algorithm 3 fixes the vertices whose shortest path distances from the left and right boundary vertex sets are strictly less than $K' = K/2 - 1$ to the left and right parts, respectively, where

K denotes the current K, which is an input of the current call of the oGPVS function. Note that the shortest path distance between any two vertices in $\mathcal{B}_L$ and $\mathcal{B}_R$ is at least $K-2$ due to this additional vertex fixing. So, this additional vertex fixing ensures that the vertex sets that are fixed to the left and right parts are disjoint and there always exists a free vertex on any path from a vertex fixed to the left part to a vertex fixed to the right part. This in turn ensures the existence of a valid vertex separator for partitioning the current graph.

This additional vertex fixing is also needed to guarantee that a K-way oVS will be obtained from RB-based partitioning of the left and right subgraphs according to Theorem 2 because of the following reasons. The above-mentioned fixing to the left part ensures that the shortest path distance between any two vertices $v_h \in \mathcal{B}_L$ and $v_i \in \mathcal{S}$ is at least $K' = K/2 - 1$ in the following $\Pi_{VS} = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$. In other words, the shortest path distance between any two vertices $v_h \in \mathcal{B}_{LL} = \mathcal{B}_L$ and $v_j \in B_{LR} = Adj(S) \cap \mathcal{V}_L$ will be at least $K/2 - 2$, where B_{LL} and B_{LR} are the left and right boundary vertex sets of left subgraph G_L, respectively. Then, G_L has a $(K/2)$-way oVS such that $B_{LL} \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ and $B_{LR} \subseteq \mathcal{V}_{K/2} \cup \mathcal{S}_{K/2-1}$, by Theorem 2. A similar discussion also holds for fixing to right part, and consequently for the right subgraph G_R. Combining these two $(K/2)$-way oVS partitions of the left and right subgraphs G_L and G_R gives a K-way oVS for the original graph G by placing the subseparator $\mathcal{S}$ (as $\mathcal{S}_{K/2}$) in between the rightmost vertex part of the left oVS and the leftmost vertex part of the right oVS. Note that having $\mathcal{B}_{LR} \subseteq \mathcal{V}_{K/2} \cup \mathcal{S}_{K/2-1}$ for the left $(K/2)$-way oVS does not violate the final K-way oVS of G, but makes consecutive subseparators adjacent via the vertices in $\mathcal{B}_{LR} \cap \mathcal{S}_{K/2-1}$. A dual discussion holds for having $\mathcal{B}_{RL} \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ for the right $(K/2)$-way oVS.

Chapter 6

Experiments

6.1 Implementation Details

Currently, existing GPVS tools such as onmetis [20] do not support fixed vertices. Instead, we utilize the Hypergraph Partitioning (HP) based GPVS formulation proposed in the citations [10, 9], since there exists a number of HP-tools such as PaToH [8], Zoltan [4] and hmetis [20] that support fixed vertices.

A hypergraph $H = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices. Every net $n_i \in \mathcal{N}$ is a subset of vertices, i.e., $n_i \subseteq \mathcal{V}$. Graph is a special instance of hypergraph such that each net connects exactly two vertices. Hypergraph partitioning problem is to partition the vertices of a hypergraph into K equal-size parts, such that the number of the nets connecting vertices in different parts is minimized. A net n_i is called a cut-net when the vertices that n_i connects are assigned to at least two different parts, whereas it is called an internal net otherwise, i.e., the vertices that n_i connects are assigned to the same part.

Since we use RB paradigm in our oGPVS solution, HP is deployed in only the bipartitioning of the graph G (or G') in Algorithm 2. We first construct the corresponding hypergraph $H = (\mathcal{V}, \mathcal{N})$ of G with the set of vertices(nodes) $\mathcal{V}$ and the set of nets $\mathcal{N}$, where each vertex v_i of G corresponds to a net n_i in H and each edge

(v_i, v_j) of G corresponds to a vertex $v_{i,j}$ in H. Each net n_i connects the vertices $v_{j,k}$ in H if and only if edge (v_j, v_k) is incident to vertex v_i in G, i.e $i = j$ or $i = k$. The objective of the bipartitioning of H using HP is to minimize the number of cut-nets while maintaining balance on the number vertices in the left and right part. After a 2-way HP on H, the resulting cut-nets correspond to the vertices in the separator, whereas the resulting internal nets correspond to the part vertices.

A vertex v_i that is fixed to the left part in G corresponds to the fixing its corresponding net n_i in H to the left part which means that n_i should be assigned as an internal net of the left part that is formed by the bipartitioning of H. Note that, the net n_i becomes an internal net of a part if and only if all of the vertices that n_i connects reside in that part, whereas it becomes a cut-net otherwise. We ensure that n_i becomes an internal net of the left part by fixing all of the vertices that is connected by n_i to the left part. A similar discussion can be made for a vertex that is fixed to the right part. The above mentioned vertex fixation scheme does not restrict the solution space of graph partitioning as seen in the following example. In G, consider a vertex v_i that is fixed to the left part and a vertex $v_h \in Adj(v_i)$ adjacent to v_i. There are two cases: v_h is a vertex that is fixed to the left part, or v_h is a free vertex. Note that v_h can not be a vertex that is fixed to the right part since vertices that are assigned to different parts can not be adjacent by both GPVS and oGPVS definition. We guarantee this not to occur by the careful selection of the number of the vertex levels to be fixed to the left and right parts. In case of v_h is a vertex that is fixed to the left part, edge (v_i, v_h) becomes an internal edge of the left part, where its corresponding vertex $v_{i,h}$ in H is also fixed to the left part by both n_i and n_h. So, the fixation of vertex $v_{i,h}$ in H does not affect the solution since v_h is a vertex that is fixed to the left part in the graph model. In case of v_h is a free vertex, $v_{i,h}$ becomes a vertex that is fixed to the left part in H, by net n_i. Fixing $v_{i,h}$ does not necessitate n_h to be an internal net in the left part, that is, n_h can be a cut-net in the bipartitioning of H which transforms to that v_h is a separator vertex in the 2-way GPVS applied on the graph. Hence, the free vertices of the graph are not affected by this fixation scheme and the solution space is not narrowed down.

6.2 Experimental Results

We have tested the performance of the oGPVS algorithm on a wide range of square sparse matrices of University of Florida (UFL) sparse matrix collection [11]. We excluded the matrices with less than 1,000 rows/columns in order to make the parallelization meaningful. We also excluded the matrices with more than 10,000,000 rows/columns since we used sequential partitioning environment. For the sake of simplicity, we considered only the matrices whose corresponding graphs are connected. There were 237 matrices in the UFL collection satisfying these properties at the time of experimentation. We tested with $K \in \{4, 8, 16, 32, 64\}$. For a specific K value, a K-way partitioning of a test matrix constitutes a partitioning instance. The partitioning instances in which $N < 100 \times K$ are discarded, as the parts would become too small to be meaningful.

We considered the graph partitioning algorithm proposed by Kahou et al, which is described in 2, as our baseline algorithm since it is the only work for solving A-to-A_{BDO} permutation problem, to our knowledge. We present the results of our experimentation in comparison with the results of this baseline algorithm. For both of these methods, we symmetrized the input matrix A with $A + A^T$ whenever A is unsymmetric. Since the first step of both methods is to find a pseudo-peripheral vertex, we ran the pseudo-peripheral node finder algorithm only once for the standard graph representation of each matrix and we used its result in both methods. For a specific K value, the partitioning process is terminated if the length of the level structure rooted at the pseudo-peripheral vertex is less than K, since the graph can not be partitioned into K parts by the baseline algorithm. So, such partitioning instances are discarded from the results of both of these methods, to make the comparison meaningful. In addition, neither methods guarantee a feasible partition, which means that there is no empty parts in the resulting partition, although the length of the level structure is larger than K. Hence, any partitioning instance, for which at least one method results in an infeasible partition, is discarded from the results of both of these methods for the sake of comparison.

We used hyperhgraph partitioning tool PaToH for the bipartitioning of a hypergraph, which is described in the previous section. As PaToH involves randomized

Table 6.1: Performance comparison in terms of load imbalance and separator size for 4-way A-to-A_{BDO} permutation

	# of	baseline algorithm		oGPVS algorithm		oGPVS vs base
problem kind	matrices	Imb.	$\|\mathcal{S}\|/N$	Imb.	$\|\mathcal{S}\|/N$	$\|\mathcal{S}_o\|/\|\mathcal{S}_b\|$
2D/3D	18	1.76%	2.08%	2.84%	1.70%	0.82
circuit simulation	7	3.34%	3.30%	2.94%	1.17%	0.35
computational fluid dynamics	23	3.76%	5.10%	4.53%	3.44%	0.67
directed graph	15	21.37%	19.45%	19.86%	12.47%	0.64
economic	6	9.64%	10.97%	21.44%	5.98%	0.54
electromagnetics	12	1.86%	2.28%	8.03%	3.15%	1.38
materials	4	6.80%	9.27%	25.10%	12.47%	1.34
model reduction	12	1.78%	2.83%	3.89%	2.30%	0.81
optimization	14	1.29%	1.32%	0.80%	0.99%	0.75
power network	3	5.30%	9.99%	4.99%	0.67%	0.07
semiconductor device	10	8.33%	10.70%	8.70%	6.17%	0.58
structural	35	3.63%	4.42%	8.18%	4.07%	0.92
theoretical/quantum chemistry	3	21.48%	27.60%	37.03%	39.65%	1.44
thermal	4	1.23%	1.38%	3.68%	1.19%	0.86
undirected graph	30	1.13%	1.23%	6.04%	0.39%	0.32

algorithms, we obtained 10 different partitions for each partitioning instance of the oGPVS method and used the geometric average of the 10 partitionings as the representative result for the oGPVS method on that particular partitioning instance. In all oGPVS partitioning instances, maximum allowable imbalance ratio , see 3.2, is set to 10%. Although the balance constraint is met in most of the partitionings, it was not feasible in some of the problems since the balancing constraint of the oGPVS problem does not exactly correspond but only relates to the balance on the nonzero counts of diagonal block D_k's and PaToH does not solve the partitioning problem optimally.

Tables 6.1, 6.2, 6.3, 6.4 and 6.5 respectively display the performance comparison of the proposed oGPVS algorithm with the baseline algorithm in terms of load imbalance and separator size for 4-, 8-, 16-, 32- and 64-way A-to-A_{BDO} permutation problem. As seen in the first column of these tables, results are categorized according to the kinds of the matrices, where each kind represents a different problem domain. In the second column, we display the number of matrices that belong to the corresponding problem kind. We included the results of the problem kinds that contain three or more

Table 6.2: Performance comparison in terms of load imbalance and separator size for 8-way A-to-A_{BDO} permutation

	# of	baseline algorithm		oGPVS algorithm		oGPVS vs base
problem kind	matrices	Imb.	$\lvert\mathcal{S}\rvert/N$	Imb.	$\lvert\mathcal{S}\rvert/N$	$\lvert\mathcal{S}_o\rvert/\lvert\mathcal{S}_b\rvert$
2D/3D	18	3.84%	4.47%	6.73%	4.12%	0.92
circuit simulation	6	4.19%	4.36%	5.69%	1.53%	0.35
computational fluid dynamics	21	8.72%	10.68%	12.56%	7.43%	0.70
directed graph	11	63.89%	29.89%	63.40%	28.51%	0.95
economic	6	23.01%	24.33%	65.40%	19.18%	0.79
electromagnetics	10	3.97%	5.52%	10.35%	4.57%	0.83
model reduction	12	5.16%	6.20%	9.63%	5.81%	0.94
optimization	11	1.17%	1.16%	1.20%	1.01%	0.87
power network	3	15.65%	20.89%	9.92%	1.77%	0.08
semiconductor device	10	14.21%	23.87%	39.18%	22.20%	0.93
structural	31	7.76%	10.35%	18.80%	9.41%	0.91
thermal	4	2.92%	2.98%	8.05%	2.75%	0.92
undirected graph	29	2.46%	2.31%	11.29%	0.77%	0.33

Table 6.3: Performance comparison in terms of load imbalance and separator size for 16-way A-to-A_{BDO} permutation

	# of	baseline algorithm		oGPVS algorithm		oGPVS vs base
problem kind	matrices	Imb.	$\lvert\mathcal{S}\rvert/N$	Imb.	$\lvert\mathcal{S}\rvert/N$	$\lvert\mathcal{S}_o\rvert/\lvert\mathcal{S}_b\rvert$
2D/3D	17	7.81%	8.47%	12.12%	7.87%	0.93
circuit simulation	5	4.01%	5.20%	8.26%	2.20%	0.42
computational fluid dynamics	19	15.03%	18.23%	27.94%	15.36%	0.84
directed graph	8	165.99%	35.75%	113.61%	27.91%	0.78
electromagnetics	8	7.63%	10.97%	15.48%	8.79%	0.80
model reduction	11	9.96%	11.91%	19.79%	10.96%	0.92
optimization	11	2.33%	2.43%	2.20%	2.16%	0.89
power network	3	44.91%	41.49%	18.31%	8.22%	0.20
semiconductor device	8	31.53%	41.73%	84.33%	41.38%	0.99
structural	25	12.79%	15.67%	26.39%	14.64%	0.93
thermal	4	6.13%	6.31%	12.72%	5.95%	0.94
undirected graph	29	6.75%	4.20%	17.75%	1.51%	0.36

Table 6.4: Performance comparison in terms of load imbalance and separator size for 32-way A-to-A_{BDO} permutation

	# of	baseline algorithm		oGPVS algorithm		oGPVS vs base
problem kind	matrices	Imb.	$\lvert\mathcal{S}\rvert/N$	Imb.	$\lvert\mathcal{S}\rvert/N$	$\lvert\mathcal{S}_o\rvert/\lvert\mathcal{S}_b\rvert$
2D/3D	15	11.53%	14.01%	22.37%	15.61%	1.11
circuit simulation	5	8.81%	10.59%	12.75%	4.80%	0.45
computational fluid dynamics	11	18.54%	22.73%	28.35%	26.87%	1.18
directed graph	3	105.69%	36.87%	82.34%	23.33%	0.63
electromagnetics	4	11.84%	10.95%	14.46%	11.87%	1.08
model reduction	8	13.91%	14.50%	28.66%	14.85%	1.02
optimization	10	4.61%	4.29%	5.18%	4.00%	0.93
structural	16	18.79%	22.40%	30.39%	19.98%	0.89
thermal	4	11.97%	12.86%	22.97%	13.08%	1.02
undirected graph	21	4.66%	3.32%	12.31%	1.21%	0.36

Table 6.5: Performance comparison in terms of load imbalance and separator size for 64-way A-to-A_{BDO} permutation

	# of	baseline algorithm		oGPVS algorithm		oGPVS vs base
problem kind	matrices	Imb.	$\lvert\mathcal{S}\rvert/N$	Imb.	$\lvert\mathcal{S}\rvert/N$	$\lvert\mathcal{S}_o\rvert/\lvert\mathcal{S}_b\rvert$
2D/3D	8	13.34%	14.23%	20.01%	14.91%	1.05
circuit simulation	3	10.57%	11.84%	12.18%	7.07%	0.60
computational fluid dynamics	5	21.16%	22.93%	36.39%	34.45%	1.50
model reduction	5	16.27%	18.67%	53.20%	19.28%	1.03
optimization	9	6.47%	6.34%	9.82%	6.92%	1.09
structural	6	32.63%	34.46%	48.74%	33.75%	0.98
undirected graph	20	9.41%	5.72%	15.29%	2.38%	0.42

matrices. The third and fourth columns display the results of the baseline algorithm, whereas fifth and sixth columns display the results of the oGPVS algorithm. The ratio of separator size over the number of vertices, i.e., $|\mathcal{S}|/N$, is given in the third and fifth columns of the baseline and oGPVS algorithms, respectively. The imbalance ratio is given in the fourth and sixth columns of the baseline and oGPVS algorithms, respectively. The seventh column gives the ratio of the separator size of the oGPVS algorithm over the separator size of the baseline algorithm. The results given in the third to seventh columns are the averages of the results of the actual instances over the matrices that belong to the corresponding problem kind. The results in the third to sixth columns are displayed as percentages.

As seen from Tables 6.1, 6.2, 6.3, 6.4 and 6.5, the oGPVS algorithm performs much better than the baseline algorithm for all specific K values by finding the separator size much smaller than the baseline algorithm does for the problem kinds such as circuit simulation, power network and undirected graph. Especially for the power network problem, we manage the separator size being equal to 7%, 8% and 20% of the separator size that the baseline algorithm finds, respectively for $K = 4$, $K = 8$ and $K = 16$. In 4-way partitionings, we perform better than the baseline algorithm in 12 problems, whereas we perform worse in only 3 problems. In 8- and 16-way partitionings, our results are better than the results of the baseline algorithm in all of the problem kinds. In 32-way partitionings, we perform much better than the baseline algorithm for the problem kinds such as circuit simulation, directed graph and undirected graph, where as we perform nearly same with the baseline algorithm for the other problems. The performance results of 64-way partitionings are similar to the performance results of the 32-way partitionings with respect to problem kind categorization.

We provide Table 6.6 for the average results for each specific $K \in \{4, 8, 16, 32, 64\}$ value. As seen from the table, the oGPVS algorithm reduces the separator sizes by 35%, 30%, 29%, 23% and 22%, in average, for 4-, 8-, 16-, 32- and 64-way partitionings, respectively. As seen in all of the tables given so far, the baseline algorithm is better than the oGPVS algorithm in terms of load imbalance ratio in almost all of the partitioning instances. However, it is preferable to have less communication with more imbalance than the case of more communication with less imbalance for parallel systems. Hence, load imbalance metric can be considered as sacrificable when the

Table 6.6: Overall performance comparison in terms of load imbalance and separator size A-to-A_{BDO} permutation

	# of	baseline algorithm		oGPVS algorithm		oGPVS vs base
K	matrices	Imb.	$\lvert\mathcal{S}\rvert/N$	Imb.	$\lvert\mathcal{S}\rvert/N$	$\lvert\mathcal{S}_o\rvert/\lvert\mathcal{S}_b\rvert$
4	205	2,99%	3,61%	5,74%	2,32%	0.65
8	183	6,00%	6,66%	12,65%	4,65%	0.70
16	155	10,22%	9,74%	18,58%	6,94%	0.71
32	106	10,73%	10,41%	18,73%	8,05%	0.77
64	63	11,25%	9,75%	18,68%	7,59%	0.78

separator size metric apts to improve.

We provide Table 6.7 to show how the pseudo-peripheral vertices affect the performance of the baseline and oGPVS algorithms. Pseudo-peripheral node finder algorithm, which is used as the first step of the both partitioning methods, involves randomization since it selects the initial node arbitrarily. The quality of the resulting partitions of these algorithms, especially for the baseline algorithm, depends on the proper selection of the pseudo-peripheral vertices. For example, the baseline algorithm constructs a level structure rooted at the pseudo-peripheral vertex and a long level structure is considered as better than a short one. In the oGPVS algorithm, we fix the vertices whose shortest path distances to pseudo-peripheral vertices are less than $K/2-1$, where K denotes the current input of the oGPVS function, and we select the separator vertices from the free vertices. Hence, the oGPVS algorithm also depends on these pseudo-peripheral vertices in terms of the solution space. For this experimentation, we ran pseudo-peripheral node finder algorithm 8 times for each matrix and we recorded the length of the level structure rooted at the resulting pseudo-peripheral vertex at each time. Then, we selected the pseudo-peripheral vertex leading to the longest level structure among the first some number of the results of 8 runs. As seen in Table 6.7, we choose this number to be in $\{1, 3, 5, 8\}$. For the first section of this table (# of pseudo-per.s is 1), we directly used the pseudo-peripheral vertex returned by the first run of pseudo-peripheral node finder algorithm. For the second section of this table (# of pseudo-per.s is 3), we selected and used the pseudo-peripheral vertex that gives the longest level structure among the first three runs of pseudo-peripheral node finder algorithm. For the other sections of this table, the selection process of

Table 6.7: Performance dependency of the algorithms to the pseudo-peripheral vertex

# of		baseline algorithm		oGPVS algorithm		oGPVS vs base
pseudo-per.s	K	Imb.	$\|\mathcal{S}\|/N$	Imb.	$\|\mathcal{S}\|/N$	$\|\mathcal{S}_o\|/\|\mathcal{S}_b\|$
	4	2,99%	3,61%	5,74%	2,32%	0,65
	8	6,00%	6,66%	12,65%	4,65%	0,70
1	16	10,22%	9,74%	18,58%	6,94%	0,71
	32	10,73%	10,41%	18,73%	8,05%	0,77
	64	11,25%	9,75%	18,68%	7,59%	0,78
	4	2,98%	3,63%	5,85%	2,39%	0,66
	8	6,05%	6,60%	12,66%	4,64%	0,70
3	16	10,02%	9,13%	17,02%	6,41%	0,70
	32	10,42%	10,03%	18,22%	7,63%	0,76
	64	10,75%	9,58%	18,43%	7,41%	0,77
	4	2,98%	3,65%	5,83%	2,36%	0,65
	8	6,18%	6,74%	13,05%	4,79%	0,71
5	16	10,32%	9,71%	18,73%	6,93%	0,71
	32	10,89%	10,42%	19,36%	8,01%	0,77
	64	10,53%	9,13%	17,42%	6,91%	0,76
	4	3,00%	3,65%	5,89%	2,33%	0,65
	8	6,02%	6,61%	12,85%	4,65%	0,70
8	16	10,25%	9,50%	18,47%	6,73%	0,71
	32	10,38%	9,86%	17,88%	7,44%	0,75
	64	10,58%	9,32%	18,09%	7,22%	0,77

Table 6.8: Performance comparison in terms of the coarsening algorithm used in PaToH

		baseline algorithm		oGPVS algorithm		oGPVS vs base
coarsening alg.	K	Imb.	$\|\mathcal{S}\|/N$	Imb.	$\|\mathcal{S}\|/N$	$\|\mathcal{S}_o\|/\|\mathcal{S}_b\|$
HCM	4	3.12%	3.76%	5.79%	2.98%	0.79
	8	6.01%	6.65%	11.63%	5.56%	0.84
	16	9.38%	8.72%	16.19%	7.29%	0.84
	32	11.05%	10.60%	18.97%	9.66%	0.91
	64	11.08%	9.68%	18.24%	9.34%	0.96
ABS	4	3.06%	3.71%	5.64%	2.85%	0.77
	8	6.10%	6.75%	12.53%	5.65%	0.84
	16	9.31%	8.70%	16.48%	7.05%	0.81
	32	9.98%	9.57%	17.50%	8.46%	0.88
	64	10.91%	9.44%	16.21%	8.34%	0.88
GCM	4	2.99%	3.62%	5.80%	2.37%	0.65
	8	6.09%	6.62%	12.52%	4.65%	0.70
	16	9.68%	9.18%	18.23%	6.65%	0.72
	32	10.33%	10.31%	18.15%	7.78%	0.75
	64	12.99%	12.45%	21.60%	10.90%	0.88
SHCM	4	2,99%	3,61%	5,74%	2,32%	0.65
	8	6,00%	6,66%	12,65%	4,65%	0.70
	16	10,22%	9,74%	18,58%	6,94%	0.71
	32	10,73%	10,41%	18,73%	8,05%	0.77
	64	11,25%	9,75%	18,68%	7,59%	0.78

pseudo-peripheral vertex is similar to the second section of the table except of that first five and eight runs are used instead of first three runs. As seen in the table, the number of the pseudo-peripheral node finder algorithm runs does not have a significant effect on the average performance. This means that only a single of the pseudo-peripheral node finder algorithm returns a good-enough pseudo-peripheral vertex for a good partitioning.

We used PaToH with default parameters except the coarsening algorithm. In the results so far, we used Scaled Heavy Connectivity Matching Algorithm as the coarsening algorithm (see PaToH manual [8]. We provide Table 6.8 for the average results of the partitioning instances using different coarsening algorithms. We provide the results for the following coarsening algorithms: Heavy Connectivity Matching (HCM), Absorption Matching (ABS), Greedy Cut Matching (GCM) and Scaled Heavy Connectivity Matching (SHCM). As seen in the table, SHCM outperforms the others.

Chapter 7

Conclusion and Future Work

Graph/hypergraph partitiong is an important tool to partition the workload among the processors for efficient parallel systems. In general, vertices represent the data and computation and the (hyper)edges represent the communication among the data. Then, the partition of the vertices, which is the output of the graph partitioning process, is used to distribute the data and computations associated with each part to a unique processor. For an efficient parallelism, communication among the processors should be minimized while maintaining a balance on the workload of each processor. Hence, the partitioning objective is to minimize the number of (hyper)edges connecting different parts since such a (hyper)edge corresponds to one unit of communication between the corresponding processors. The constraint of the partitioning is to maintain a balance on the part weights since a part weight generally corresponds to the workload of the corresponding processor.

In this thesis, we have solved A-to-A_{BDO} permutation problem, where A_{BDO} stands for the block diagonal form (BDO) of A with overlap. In a BDO form of A, which is to be used in the parallelization of the multiplicative schwarz preconditioner, consecutive diagonal blocks may overlap. The permutation objective is to minimize the number of the rows/columns in the overlaps while maintaining a balance on the nonzero counts of the diagonal blocks. We introduced the ordered GPVS (oGPVS) problem, which is a variant of the Graph Partitioning by Vertex Separator (GPVS) problem, in order to solve the A-to-A_{BDO} permutation problem. The standard GPVS

problem is to partition the vertices into K vertex parts and one separator such that the removal of the vertices in the separator from the graph disconnects the graph into at least K connected subgraphs, where the partitioning objective is to minimize the number of vertices in the separator while maintaining a balance on the weights of K parts. However, the oGPVS problem is to partition the vertices into K vertex parts and $K-1$ subseparators such that the removal of the vertices in the separator from the graph disconnects the graph into at least K connected subgraph, subseparators and vertex parts exhibit an order and each subseparator connects its left and right vertex parts as well as its left and right subseparators.

We have proposed a recursive bipartitioning (RB) based algorithm, namely oGPVS algorithm, to solve oGPVS problem. In order to guarantee that the resulting partition is a correct solution of K-way oGPVS problem, we adopted a novel vertex fixation scheme in RB paradigm. The oGPVS algorithm performs better than Kahou's partitioning algorithm, which is considered as our baseline algorithm, by finding the average separator size 65%-78% of the average separator size of the baseline algorithm. This result is valuable because a small separator size corresponds to a small number of rows/columns in overlaps, which means that the communication overhead of the parallelization and the number of iterations for convergence of the solver will be both small.

As the future work, we plan to find a graph partitioning model for the A-to-A_{BDO} problem with unsymmetric permutation, where A is an unsymmetric matrix. For this problem, the standard graph model will not be adequate, hence a bipartite graph or hypergraph model is planned to be used.

Bibliography

[1] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *VLSI Journal*, 19(1–2):1–81, 1995.

[2] C. Aykanat, B. B. Cambazoglu, F. Findik, and T. Kurc. Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids. *Journal of Parallel and Distributed Computing*, 67:77–99, 2006.

[3] M. Benzi and B. Uar. Block triangular preconditioners for m-matrices and markov chains. *Electronic Transactions on Numerical Analysis*, 26:209–227, 2007.

[4] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, U. C. C. Vaughan, D. Bozdag, W. Mitchell, and J. Teresco. Zoltan 3.0: Parallel partitioning, load-balancing, and data management services; users guide. Sandia National Laboratories, Albuquerque, NM, 2007.

[5] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is np-hard. *Information Processing Letters*, 42:153–159, 1992.

[6] B. B. Cambazoglu and C. Aykanat. Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids. *IEEE Transactions on Parallel and Distributed Systems*, 18(1):3–16, Jan 2007.

[7] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy, and L. A. Riesen. A repartitioning hypergraph model for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 69(8):711–724, 2009.

[8] U. V. Çatalyürek and C. Aykanat. Patoh: A multilevel hypergraph partitioning tool, version 3.0. Computer Engineering Department, Bilkent University, 1999.

[9] U. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based sparse matrix ordering. In *Second International Workshop on Combinatorial Scientific Computing (CSC05)*, CERFACS, Toulouse, France, June 2005.

[10] U. V. Çatalyürek, C. Aykanat, and E. Kayaaslan. Hypergraph partitioning-based fill-reducing ordering. *SIAM Journal on Scientific Computing*, 2001.

[11] T. Davis. University of florida sparse matrix collection. In *NA Digest*, 1997.

[12] A. L. Dulmage and N. S. Mendelsohn. Two algorithms for bipartite graphs. *Journal of the Society for Industrial and Applied Mathematics*, 11:183–194, 1963.

[13] C. Farhat and F.-X. Roux. An unconventional domain decomposition method for an efficient parallel solution of large-scale finite element systems. *SIAM Journal on Scientific and Statistical Computing*, 13:379–396, 1992.

[14] A. George. An automatic one-way dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 17:740–751, 1980.

[15] A. George and J. W. H. Liu. An implementation of a pseudo-peripheral node finder. *Applied Numerical Mathematics*, 57(11–12):1197–1213, 2007.

[16] B. Hendrickson and R. Leland. The chaco users guide, version 2.0. Sandia National Laboratories, Alburquerque, NM, 87185, 1995.

[17] G. A. A. Kahou, L. Grigori, and M. Sosonkina. A partitioning algorithm for blockdiagonal matrices with overlap. *Parallel Computing*, 34:332–344, 2008.

[18] G. A. A. Kahou and E. Kamgnia. Parallel implementation of an explicit formulation of the multiplicative schwarz preconditioner. In *CdROM Proceedings of IMACS05*, 2005.

[19] G. A. A. Kahou, E. Kamgnia, and B. Philippe. An explicit formulation of the multiplicative schwarz preconditioner. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):284–295, 1979.

[20] G. Karypis and V. Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[21] X. S. Li, M. Shao, I. Yamazaki, , and E. G. Ng. Factorization-based sparse solvers and preconditioners. In *Journal of Physics: Conference Series*, volume 180, page 012015, 2009.

[22] J. W. H. Liu. A graph partitioning algorithm by node separators. *ACM Trans. Math. Softw*, 15:198–219, 1989.

[23] I. Moulitsas and G. Karypis. Partitioning algorithms for simultaneously balancing iterative and direct methods, tech. rep. 04-014. Department of Computer Science and Engineering, University of Minnesota, 2004.

[24] F. Pellegrini. Scotch 5.1 users guide. Laboratoire Bordelais de Recherche en Informatique (LaBRI), 2008.

[25] A. Pinar and C. Aykanat. Fast optimal load balancing algorithms for 1d partitioning. *Journal of Parallel and Distributed Computing*, 64:974–996, 2004.

[26] A. Pinar and B. Hendrickson. Apartitioning for complex objectives. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium, IPDPS 01*, Washington, DC, USA, 2001.

[27] M. Wimmer and K. Richter. Optimal block-tridiagonalization of matrices for coherent charge transport. *Journal of Computational Physics*, 228(23):8548–8565, 2009.

[28] I. Yamazaki, X. S. Li, and E. G. Ng. Partitioning, load balancing, and matrix ordering in a parallel hybrid solver, presentation at siam conference on parallel processing for scientific computing (pp10). 2010.